Oracle® Cloud

Developing SOA Applications with Oracle SOA Suite

12*c* (12.1.3)

E60643-03

October 2016

Documentation for developers that describes how to design, secure, test, and deploy Oracle Service-Oriented Architecture (SOA) composite applications consisting of service and reference binding components and Oracle BPEL process, human task, business rule, Oracle Mediator, and spring service components. Includes additional information on designing transformations and business events and acting upon human tasks during runtime in Oracle BPM Worklist.

ORACLE

Oracle Cloud Developing SOA Applications with Oracle SOA Suite, 12*c* (12.1.3)

E60643-03

Primary Author: Oracle Corporation

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

Contents

3 Managing Shared Data with the Design-Time MDS Repository

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 Correlation Sets and Message Aggregation

15 Using Events and Timeouts in BPEL Processes

16 Coordinating Master and Detail Processes

17 Using the Notification Service

18 Using Oracle BPEL Process Manager Sensors and Analytics

Part III Using the Oracle Mediator Service Component

19 Getting Started with Oracle Mediator

26 Using Declarative Components and Task Flows

29 Configuring Human Tasks

30 Designing Task Forms for Human Tasks

37 Integrating Enterprise JavaBeans with Composite Applications

38 Using Direct Binding to Invoke Composite Services

Part VII Sharing Functionality Across Service Components

39 Oracle SOA Suite Templates and Reusable Subprocesses

40 Creating Transformations with the XSLT Map Editor

41 Creating Transformations with the XQuery Mapper

42 Using Business Events and the Event Delivery Network

47 Deploying SOA Composite Applications

49 Debugging and Auditing SOA Composite Applications

50 Automating Testing of SOA Composite Applications

53 Defining Composite Sensors

54 Creating Dynamic Business Processes

55 Integrating the Spring Framework in SOA Composite Applications

B XPath Extension Functions

F Interfaces Implemented By Rules Dictionary Editor Task Flow

G Oracle SOA Suite Configuration Properties Road Map

H Working with Large Schemas in the XSLT Editor

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

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc`.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

Related Documents

For more information, see the following Oracle resources:

- *User's Guide for Oracle B2B*
- *Healthcare Integration User's Guide for Oracle SOA Suite*
- *Monitoring Business Activity with Oracle BAM*
- *Understanding Technology Adapters*
- *Designing Business Rules with Oracle Business Process Management*
- *Rules Language Reference for Oracle Business Process Management*

- *Administering Oracle SOA Suite and Oracle Business Process Management Suite*
- *Tuning Performance*
- *Enterprise Deployment Guide for Oracle SOA Suite*
- *High Availability Guide*
- *WLST Command Reference for WebLogic Server*

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.

What's New in This Guide

This preface introduces the new and changed features of Oracle SOA Suite and other significant changes that are described in this guide, and provides pointers to additional information. This document is the new edition of the formerly titled *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

For a list of known issues (release notes), see `http://www.oracle.com/technetwork/middleware/docs/soa-aiafp-knownissuesindex-364630.html`.

New and Changed Features for 12*c* (12.1.3)

For Oracle SOA Suite 12*c* (12.1.3), this guide has been updated to include the following new and changed development features:

- Reorganization of SOA composite application directories and files in the Applications window. See What Happens When You Create a SOA Application and Project.
- Support for adding SOA composite application descriptions that are displayed when you place your cursor over the **TODO Tasks** icon above the composite. See Adding Descriptions to SOA Composite Applications.
- Support for renaming, deleting, and moving components and their artifacts in the SOA Composite Editor and Applications window. See Renaming_ Deleting_ and Moving Components and Artifacts.
- Support for managing shared data with the design-time Oracle Metadata Services Repository (MDS Repository). See Managing Shared Data with the Design-Time .
- Reorganization of BPEL process directories and files in the Applications window. See How to Add a BPEL Process Service Component.
- Support for editing BPEL process activities in the Property Inspector. How to Edit BPEL Activities in the Property Inspector.
- Support for adding BPEL process service component descriptions that are displayed when you place your cursor over the **TODO Tasks** icon above the BPEL process. See How to Add a Description of Actions to BPEL Process Activities.
- Support for translating messages between native XSD format and XML format in a BPEL process translate activity. See Translating Between Native Data and XML.
- Support for creating BPEL process correlations sets with the Correlation Wizard. How to Create a Correlation Set with the Correlation Wizard.

- Support for designing a fault policy with the Fault Policy wizard. See How to Design a Fault Policy for Automated Fault Recovery with the Fault Policy Wizard.
- Support for executing a business process without a transaction. See Executing a Business Process Without a Transaction.
- Support for invoking an Oracle Enterprise Scheduler job in a BPEL process. Invoking an Oracle Enterprise Scheduler Job in a BPEL Process.
- Support for BPEL process analytics (business indicators and measurements). See Configuring BPEL Process Analytics.
- Support for integrating Representational State Transfer (REST) operations in SOA composite applications. See Integrating REST Operations in SOA Composite Applications.
- Support for using Oracle SOA Suite templates in SOA composite applications, service components, and BPEL scope activities. See Oracle SOA Suite Templates and Reusable Subprocesses .
- Support for creating and reusing standalone and inline BPEL subprocesses within other processes. See Oracle SOA Suite Templates and Reusable Subprocesses .
- Support for a standard JMS-based messaging infrastructure in the Oracle SOA Suite event delivery network (EDN). See EDN Integration with .
- Support for obfuscating certain fields (for example, SSNs) to prevent this data from appearing in clear text. See Encrypting and Decrypting Specific Fields of Messages.
- Support for using the Maven plug-in to build and manage SOA composite application projects. See Using the Development Maven Plug-In.
- Support debugging SOA composite applications with the SOA debugger in Oracle JDeveloper. See Debugging and Auditing SOA Composite Applications .
- Support for creating test suites and their test cases with the Create SOA Composite Test wizard. See Creating Test Suites and Test Cases with the Create Composite Test Wizard.
- Support for running test suites from Oracle JDeveloper. See Deploying and Running a Test Suite.
- Support for creating, updating, and deleting composite sensors during runtime from Oracle SOA Composer. Creating and Managing Composite Sensors During Runtime from Oracle SOA Composer.
- Support for running the Native Format Builder wizard outside of the Adapter Configuration wizard. See Using the Native Format Builder Wizard Outside of Adapter Configuration.

Other Significant Changes in this Document for 12*c* (12.1.3)

For Oracle SOA Suite 12*c* (12.1.3), this guide has been updated in the following ways:

- The Oracle User Messaging Service information that appeared in the 11*g* Release 1 (11.1.1) version of this guide has been moved to *Developing Applications with Oracle User Messaging Service*.

- The Oracle Business Activity Monitoring (BAM) information that appeared in the 11*g* Release 1 (11.1.1) version of this guide has been moved to *Monitoring Business Activity with Oracle BAM*.
- The Configuring Task List Portlets chapter has been removed. Oracle WebCenter is not supported in this release of Oracle SOA Suite.

Part I

Getting Started with 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 service-oriented architecture (SOA) and Oracle SOA Suite, standards used by Oracle SOA Suite to enable SOA, SOA composite application architecture and runtime behavior, approaches to designing SOA composite applications, and where to go to learn more about Oracle SOA Suite.

This chapter includes the following sections:

- Introduction Oracle SOA Suite
- Getting Started with Oracle SOA Suite
- Setting Accessibility Options

Differences Between Using this Component in the Cloud and On-Premises Environments

There may be differences between using this component in the cloud and on-premises environments that impact the information described in this guide.

For information about differences, see Differences Between the Cloud and On-Premises Environments and Known Issues for Oracle SOA Cloud Service.

Introduction to Oracle SOA Suite

This section provides an overview of service-oriented architecture and standards, Oracle SOA Suite capabilities, service component architecture, runtime behavior, and design-time approaches.

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.

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.

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
- Human interaction
- Events framework
- Business activity monitoring

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.

- XQuery Transformations (XQuery)

  Queries and transforms collections of structured and unstructured data, typically in the form of XML.

- 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 Service Definition 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.

- Representational State Transfer (REST)

  Provides an architecture for designing network applications. RESTful applications use HTTP requests to post data (create and update), get data (for example, make queries), and delete data. REST provides an alternative to using web services.

- JavaScript Object Notation (JSON)

Provides a language for representing simple data structures and associative arrays called objects. JSON is a standard designed for human-readable data interchange. JSON is derived from the JavaScript scripting language.

- Web Application Description Language (WADL)

  Provides a readable XML description of HTTP-based web applications (typically REST web services). WADL simplifies the reuse of web services based on the existing HTTP architecture of the web.

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, 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

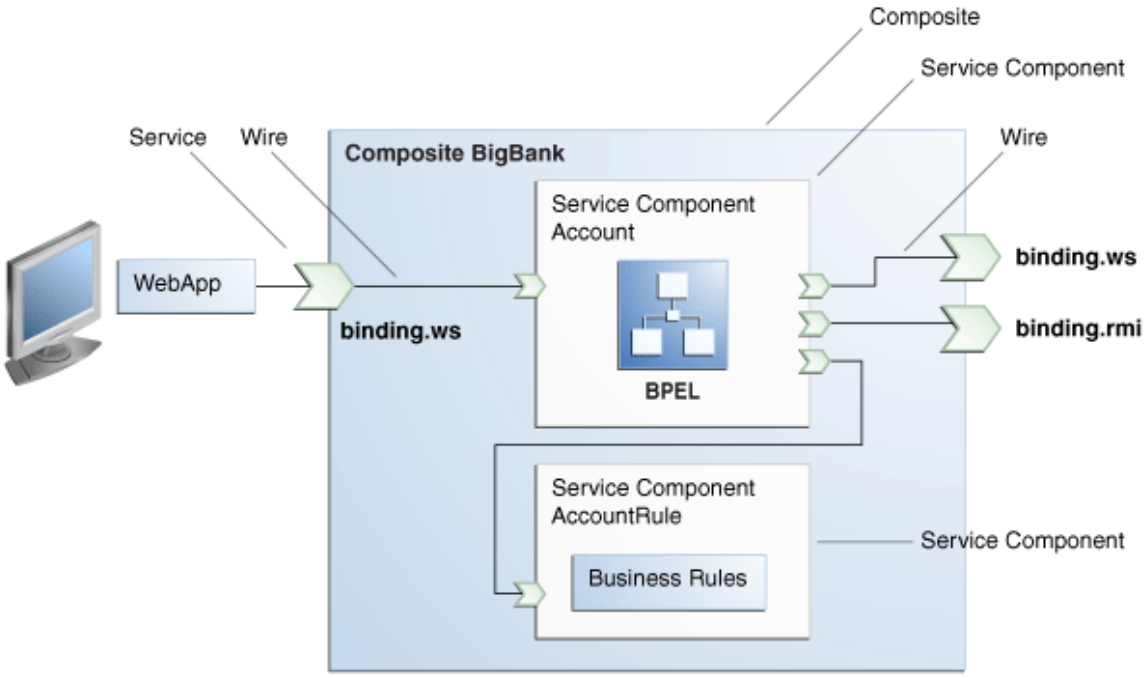
  Service components can be easily maintained and debugged when an issue is encountered.

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



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.

For more information about service components, see Adding Service Components.

Binding Components

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

- Services

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

  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, Oracle User Messaging Service, LDAP servers, Oracle Coherence cache, 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, socket adapter, Oracle User Messaging Service adapter, LDAP adapter, Oracle Coherence adapter, and third-party adapter.
Oracle B2B	Use for browsing B2B metadata in the Oracle Metadata Services Repository (MDS Repository) and selecting document definitions.
Oracle Healthcare	Use for sending and receiving messages to and from a healthcare system.
ADF-BC service	Use for connecting Oracle Application Development Framework (ADF) applications using SDO with the SOA platform.
Oracle E-Business Suite	Use for integrating the Oracle E-Business Suite adapter with Oracle applications.
BAM 11g adapter	Use for integrating Java EE applications with Oracle BAM 11*g* server to send data, and also use as a reference binding component in a SOA composite application. **Note**: This adapter can only connect to an Oracle BAM 11*g* server.
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.

Table 1-1 (Cont.) Binding Components Provided by Oracle SOA Suite

Binding Components	Description
HTTP binding	Use to integrate SOA composite applications with HTTP binding.
REST service	Use to integrate REST services with SOA composite applications and REST-enable SOA composite applications.
Oracle Managed File Transfer (MFT)	Use to transfer files to and from many endpoint types, such as remote and embedded FTP or sFTP servers; directories; and SOAP web service, Oracle SOA Suite, Oracle Service Bus, Oracle B2B, Oracle Healthcare, and Oracle Data Integrator endpoints.
Cloud adapters	The cloud adapters enable you to send and receive messages from a cloud server.

For more information about binding components, see Adding Service Binding Components and Adding Reference Binding Components.

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

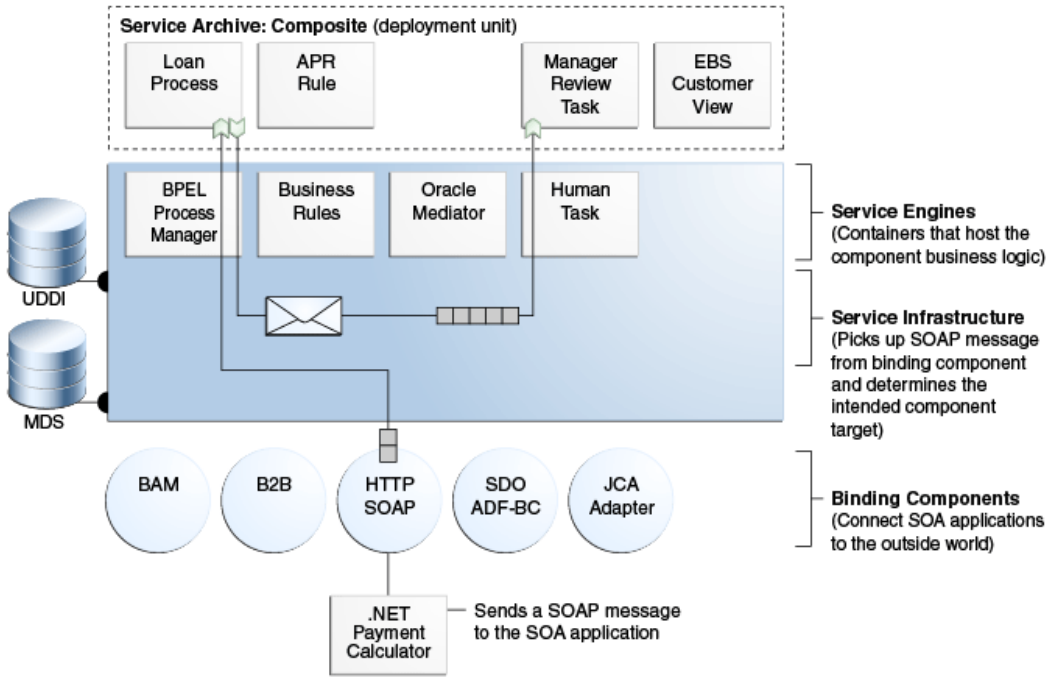
For more information about wires, see Adding Wires.

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 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 1-2.

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

Part	Description	Example of Use in Figure 1-2	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 *reference* binding component in Figure 1-2 is the Loan Process application.	Service Components

Table 1-2 (Cont.) Introduction to a SOA Composite Application Using SCA Technologies

Part	Description	Example of Use in Figure 1-2	See Section
Service Infrastructure	Provides internal message transport	The Service Infrastructure: • Receives the message from the SOAP service binding component. • Posts the message for processing to the BPEL process service engine first and the human task service engine second.	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 process 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.	Service Engines
Universal Description, Discovery, and Integration (UDDI) and MDS	The MDS 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.	olink:SOAGSManaging Shared Data with the Design-Time
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.	Deployed Service Archives

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

Service Engines

Service engines are containers that host the business logic or processing rules of the 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 *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

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 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).

For more information, see Deploying SOA Composite Applications .

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 . 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.

Getting Started with Oracle SOA Suite

This developer's guide consists of the sections described in Table 1-3. These sections enable you to get started with developing a SOA composite application.

Table 1-3 Getting Started with Oracle SOA Suite

To Get Started with...	See...
The basic steps of composite, service and reference binding component, and service component creation in Oracle JDeveloper	Getting Started with Developing SOA Composite Applications
Using shared data with the SOA Design-Time Oracle Metadata Services Repository (MDS Repository)	Managing Shared Data with the Design-Time
Designing BPEL process service components in a composite	Using the BPEL Process Service Component
Designing Oracle Mediator service components in a composite	Using the Oracle Mediator Service Component
Designing business rule service components in a composite	Using the Business Rules Service Component
Designing human workflow service components in a composite	Using the Human Workflow Service Component

Table 1-3 (Cont.) Getting Started with Oracle SOA Suite

To Get Started with...	See...
Designing service and reference binding components in a composite	Using Binding Components
Functionality that can be shared across components, such as templates, XSLT and XQuery transformations, business events, cross references, and domain value maps	Sharing Functionality Across Service Components
Composite completion tasks such as security policy attachments, deployment, debugging, and automating composite testing	Completing Your Application
Advanced topics such as management of large documents and large numbers of instances, composite customizations, composite sensors, and the spring framework	Advanced Topics

In addition to this developer's guide, other resources are provided:

- Oracle SOA Suite samples provide access to various use case samples for Oracle SOA Suite and its components.
- *Understanding Oracle SOA Suite* describes the business challenges faced by a company and how the components of Oracle SOA Suite address these challenges from design time through runtime.

Setting Accessibility Options

Oracle SOA Suite uses both Oracle JDeveloper and Oracle SOA Composer for application development. This section describes accessibility options for both environments.

Setting Accessibility Options in Oracle JDeveloper

Oracle JDeveloper provides accessibility options, such as support for screen readers, screen magnifiers, and standard shortcut keys for keyboard navigation. You can also customize Oracle JDeveloper for better readability, including the size and color of fonts and the color and shape of objects. For information and instructions on configuring accessibility in Oracle JDeveloper, see "Oracle JDeveloper Accessibility Information" in *Developing Applications with Oracle JDeveloper*.

Setting Accessibility Options in Oracle SOA Composer and Oracle BPM Worklist

Accessibility settings help you read all components of the application. You can set accessibility options in either Oracle SOA Composer or Oracle BPM Worklist for the current instance or for all instances.

How to Set Accessibility Features Before Logging In

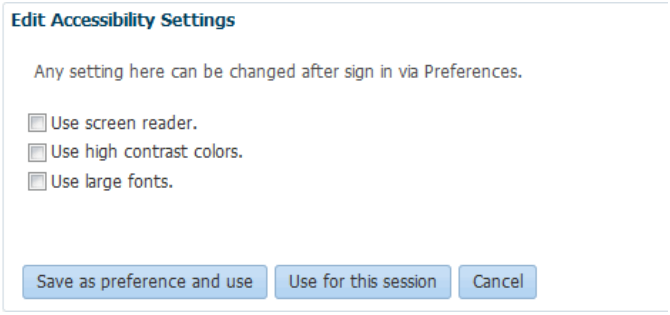
Oracle SOA Composer or Oracle BPM Worklist presents the Accessibility menu on the login page, so you can configure accessibility before you log in. These settings can be persisted for only the current session or for all sessions.

To set accessibility options before logging in:

1. Launch Oracle SOA Composer or Oracle BPM Worklist.
2. On the login page, click **Accessibility** in the top right corner.

   The Edit Accessibility Settings page appears, as shown in Figure 1-3.

   Figure 1-3 Edit Accessibility Settings Page

3. Select any of the following options:
   - Use screen reader.
   - Use high contrast colors.
   - Use large fonts.
4. To save the new settings only for this session, click **Use for this session**. To save the settings for all sessions, click **Save as preference and use**.

How to Set Accessibility Options After Logging In

Once you log in to Oracle SOA Composer or Oracle BPM Worklist, you can configure accessibility options from any page. This changes the user preferences, which are retained through all sessions until you change them again.

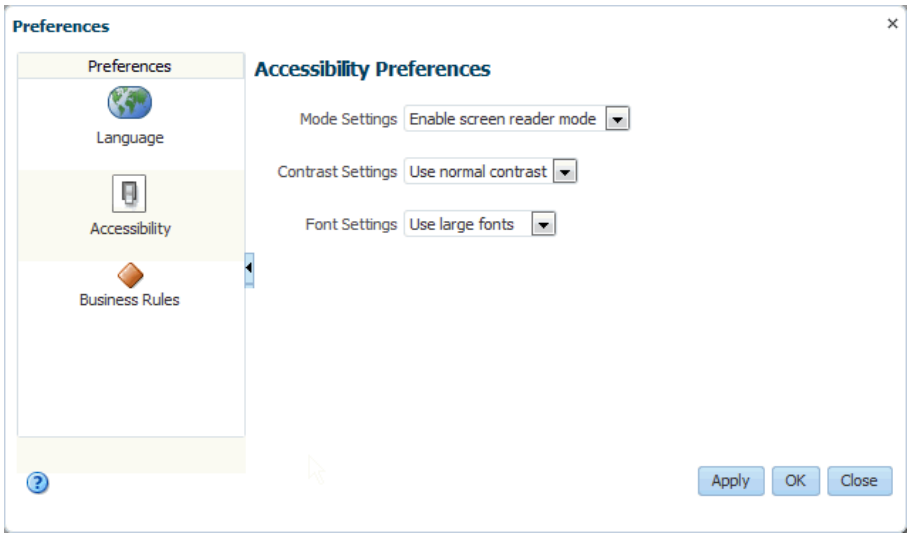
To set accessibility options after logging in:

1. Launch Oracle SOA Composer or Oracle BPM Worklist and log in.
2. From any page, select **Preferences** in the top right corner.

   The Preferences dialog appears.
3. In the Preferences column, click **Accessibility**.

   The Accessibility Preferences appear, as shown in Figure 1-4.

Figure 1-4 Preferences Dialog



4. In the **Mode Settings** field, select **Enable screen reader mode** if you use a screen reader. Select **Disable screen reader mode** if you do not use a screen reader.
5. In the **Contrast Settings** field, select **Use high contrast** to increase the contrast between objects on the console; otherwise, select **Use normal contrast**.
6. In the **Font Settings** field, select **Use large fonts** to increase the font size; otherwise, select **Use normal fonts**.
7. Click **OK**.

2

Getting Started with Developing SOA Composite Applications

This chapter describes how to use Oracle JDeveloper to create a SOA composite application. It guides you through the basic steps of composite, service and reference binding component, and service component creation, security, deployment, and testing, along with describing key issues to be aware of when designing a SOA composite application.

This chapter includes the following sections:

Creating a SOA Application

The first steps in building a new application are to assign it a name and to specify the directory in which to save source files. When you install the Oracle SOA Suite Quick Start, the Oracle SOA Suite extensions are automatically installed in Oracle JDeveloper. This differs from previous releases in which you manually imported the Oracle SOA Suite extensions into Oracle JDeveloper. For information about the Oracle SOA Suite Quick Start installation, see *Installing SOA Suite and Business Process Management Suite Quick Start for Developers*.

How to Create a SOA Application and Project

You first create an application for the SOA project.

To create a SOA application and project:

1. Start Oracle JDeveloper Studio Edition.
2. If Oracle JDeveloper is running for the first time, specify the location for the Java JDK and the user role in which to run Oracle JDeveloper. The JDK version must be later than or equal to 1.7.0_15.
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 Applications window in the upper left, click **New Application**.
Has existing applications.	From the **File** main menu: **a.** Select **New** > **Application**. The New Gallery opens, where you can select different application components to create. **b.** In the **Categories** tree, select **General** > **Applications**. **c.** In the **Items** pane, select **SOA Application**, and click **OK**. From the **Application** main menu: **a.** Select **New**. The New Gallery opens, where you can select different application components to create. **b.** In the **Categories** tree, select **General** > **Applications**. **c.** In the **Items** pane, select **SOA Application**, and click **OK**. From the **Application** menu in the Applications window. **a.** In the Applications window in the upper left, select **New Application** from the **Applications** dropdown list.

The Create SOA Application wizard is displayed.

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

Note:

Note the following application naming conventions:

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

  In a design-time environment, if you are using Oracle JDeveloper, select **Tools** > **Preferences** > **Environment** > **Encoding** > **UTF-8** to enable Unicode support. This 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` file that describes the SOA composite application, and the necessary libraries to your model project.
6. 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.

7. In the Configure SOA Settings page, click **Empty Composite** for this example, and click **Finish**. Table 2-3 describes all of the options on this page.

Table 2-3 Configure SOA Settings Page

Element	Description
Empty Composite	Creates an empty SOA composite application. This type is selected by default.
Composite With BPEL Process	Automatically opens the Create BPEL Process dialog to guide you through creation of an initial BPEL process. A BPEL process enables you to design a business process that integrates a series of business activities and services into an end-to-end process flow.
Composite With Mediator	Automatically opens the Create Mediator dialog to guide you through creation of an initial Oracle Mediator service component. Oracle Mediator enables you to route events (messages) between different components.
Composite With Human Task	Automatically opens the Create Human Task dialog to guide you through creation of an initial human task service component. A human task component enables 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. The tasks are accessed through Oracle BPM Worklist during process runtime.
Composite With Subprocess	Automatically creates a SOA composite application with a subprocess. A subprocess is a fragment of BPEL code that can be reused within a particular processor by separate processes.
Composite With Business Rule	Automatically opens the Create Business Rules dialog to guide you through creation of an initial business rule service component. A business rule enables you to design a business decision based on rules.
Composite With Spring	Automatically opens the Create Spring dialog to guide you through creation of a spring context service component. A spring context service component enables you to integrate components that use Java interfaces instead of WSDL files into SOA composite applications. You can also integrate components that use Java interfaces with components that use WSDL files in the same SOA composite application.

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

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

Figure 2-1 shows the SOA Composite Editor for a project named **OrderBookingComposite**.

Figure 2-1 New Workspace for a SOA Composite Application

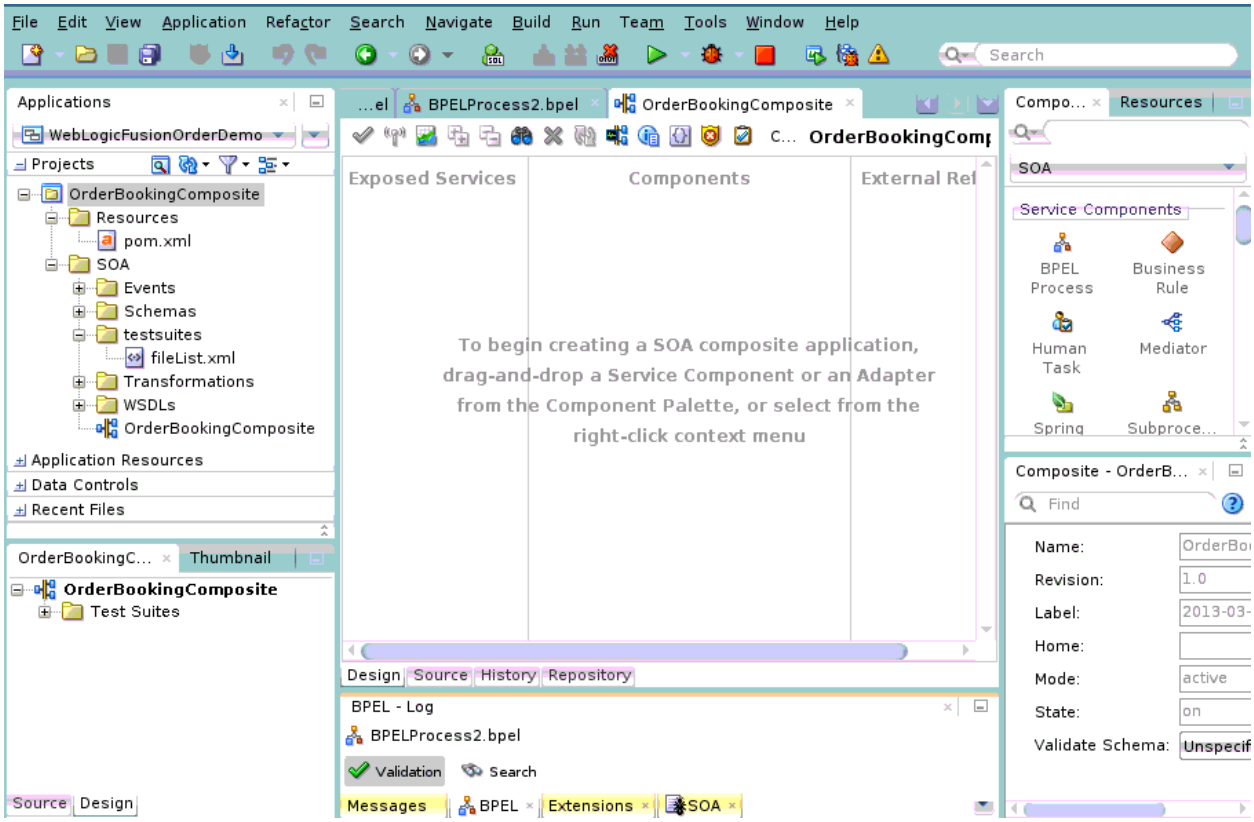


Table 2-4 describes the SOA Composite Editor.

Table 2-4 SOA Composite Editor

Element	Description
Applications Window (Upper left)	Displays the key directories and files for the specific service components included in the SOA project. You can change the structure as necessary for your environment. The only limitation is that all files must be located under the **SOA** directory. • ***Service_component_directory*** Displays a directory for the artifacts of each service component you add: A **BPEL** directory is created for BPEL processes. A **Mediators** directory is created for Oracle Mediators. A **HumanTasks** directory is created for human tasks. An **oracle/rules** directory is created for business rules. • **Events** Displays the business event files (**.edn**). • **Schemas** Displays the BPEL process schema files (**.xsd**). • **testsuites** Displays the test suite files. • **Transformations** Displays the transformation XSLT (**.xsl**) and XQuery (**.xqy**) mapper files. • **WSDLs** Displays all WSDL files (**.wsdl**). • ***composite_name*** A ***composite_name*** file is automatically created when you create a SOA project. This file describes the entire composite assembly of services, service components, references, and wires.
Structure Window (Lower left)	The Structure window provides a structural view of the data in the document currently selected in the active window.
Designer (middle)	You drag service components, services, and references from the Components window into the composite in the designer. When you drag and drop a service component into the designer, 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 for configuring the Oracle Mediator service component. For all subsequent editing sessions, you double-click these service components to re-open their editors.
Project Name (Above the designer)	Displays the project name of the SOA composite application.
Left Swimlane (Exposed Services)	The left swimlane is for services (such as web services, REST adapters, or JCA adapters) that provide an entry point to the SOA composite application.

Table 2-4 (Cont.) SOA Composite Editor

Element	Description
Right Swimlane (External References)	The right swimlane is for references that send messages to external services in the outside world, such as web services or JCA adapters.
Components Window (Upper right - Components tab)	The Components window provides the various resources that you can use in a SOA composite. It contains the following service components and adapters: • Components Displays the BPEL process, business rule, human task, Oracle Mediator, and spring components that can be dragged and dropped into the designer. • Technology Displays the JCA adapters (such as AQ, file, FTP, database, JMS, MQ, Oracle User Messaging Service, socket, LDAP server, and Coherence cache), third-party adapter, cloud adapter, Oracle BAM 11*g* binding component, Oracle Healthcare binding component, Oracle B2B binding component, EJB binding component, ADF-BC binding component, application adapters (Oracle E-Business Suite, JDE World, and SAP), direct binding component, HTTP binding component, Oracle Managed File Transfer (MFT) adapter, Representational State Transfer (REST) adapter, and web service binding component that can be dragged into the left or right swimlane.
Resources window (Upper right - Resources tab)	The Resources window provides a single dialog from which you can browse both local and remote resources. For example, you can access the following resources: • Shared data such as schemas and WSDLs from the MDS Repository. • 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. You select these resources for the SOA composite application through the WSDL Chooser 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 **Resources** at the top of this dialog to access available resources.
Log Window (Lower middle)	The Log window displays messages about application compilation, validation, and deployment.
Property Inspector (Lower right)	The Property Inspector displays properties for the selected service component, service, or reference. You can also edit BPEL activity properties and define deployment descriptor properties for a BPEL process service component. For more information, see How to Edit BPEL Activities in the Property Inspector. and How to Define Deployment Descriptor Properties in the Property Inspector.

Table 2-4 (Cont.) SOA Composite Editor

Element	Description
Application View	The Application View shows the artifacts for the SOA composite application.

The *composite_name* file (also known as the **composite.xml** file) displays as a tab in the designer and as a file in the Applications window. 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.

Adding Service Components

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

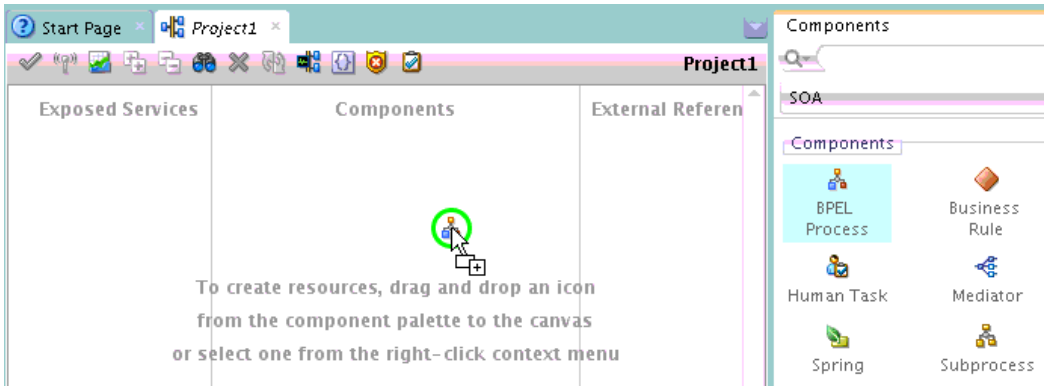
How to Add a Service Component

To add a service component:

1. At the top of the Components window, click **Components**.
2. From the **SOA** section, drag a component into the designer.

   Figure 2-2 shows a BPEL process being added to the designer.

Figure 2-2 Adding a BPEL Process to the SOA Composite Application



A specific dialog for the selected service component is displayed. Table 2-5 describes the available editors.

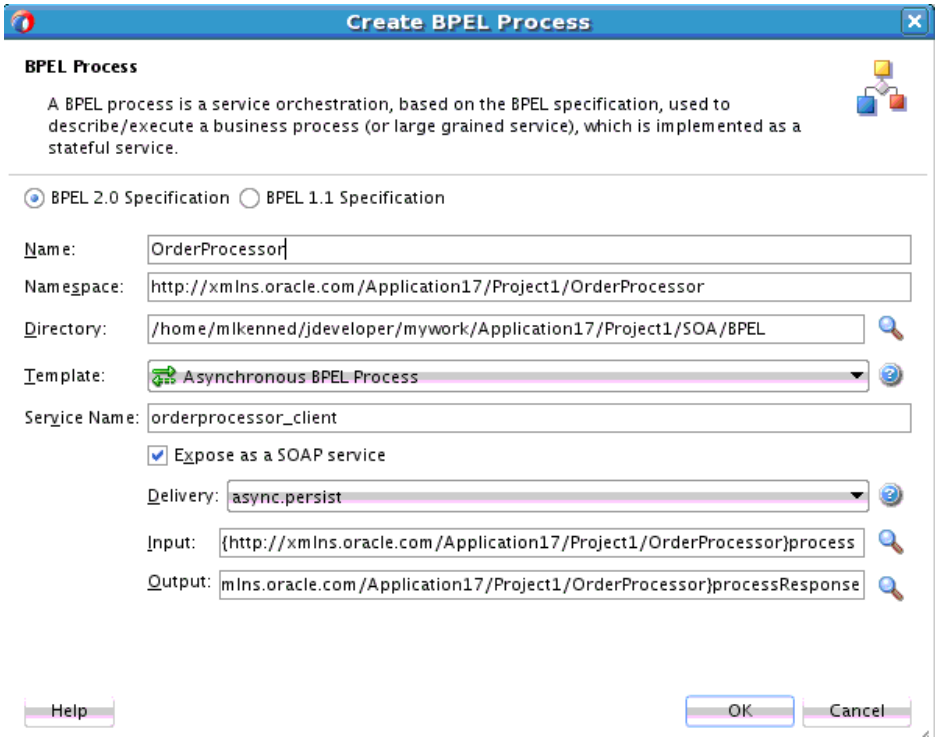
Table 2-5 Starting Service Component Editors

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 Component	Create Spring dialog to create a spring context file for integrating Java interfaces into SOA composite applications.

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

   Figure 2-3 shows the BPEL Process dialog with data entered to create the **OrderProcessor** BPEL process. The process is selected to be asynchronous. The **Expose as a SOAP Service** check box directs Oracle JDeveloper to automatically create this service component connected to an inbound SOAP web service.

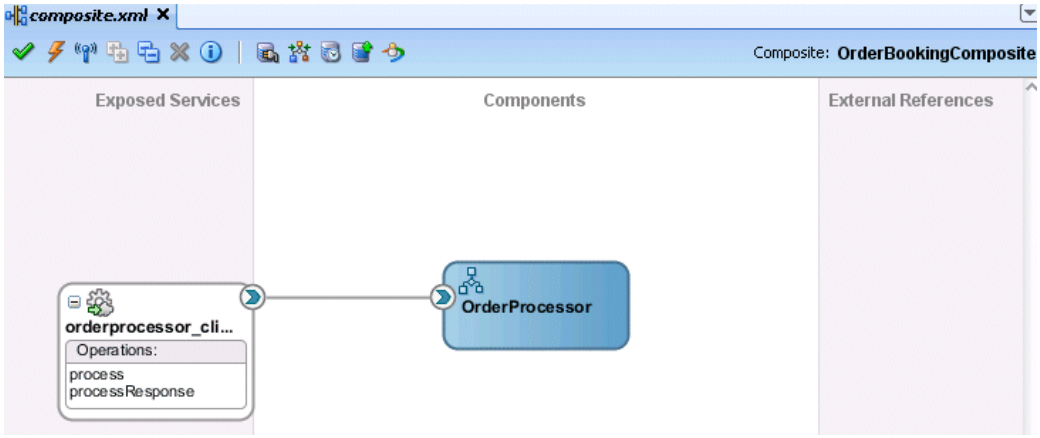
Figure 2-3 Create BPEL Process Dialog



4. Click **OK**.

Figure 2-4 shows the **OrderProcessor** BPEL process service component in the designer. 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 does not appear. You can add a service later by following the steps in How to Add a Service Binding Component.

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**.

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 Resources window to browse for service components defined in the , 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 Applications window.
- 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 . In addition, the partner link to the task can be deleted. Deleting the partner link removes the reference interface and removes the wire to the task.

How to Edit a Service Component

You edit 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 2-6.

Table 2-6 Starting SOA Service Component Wizards and Dialogs

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

2. Modify the settings for the selected service component. For help with a service component editor or designer, click **Help** or press **F1**. These editors are described in later chapters.
3. From the **File** main menu, select **Save All**.
4. In the Applications window, double-click ***composite_name*** or single-click ***composite_name*** above the designer.

   This action returns you to the .

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.

How to Add a Service Binding Component

Note:

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.

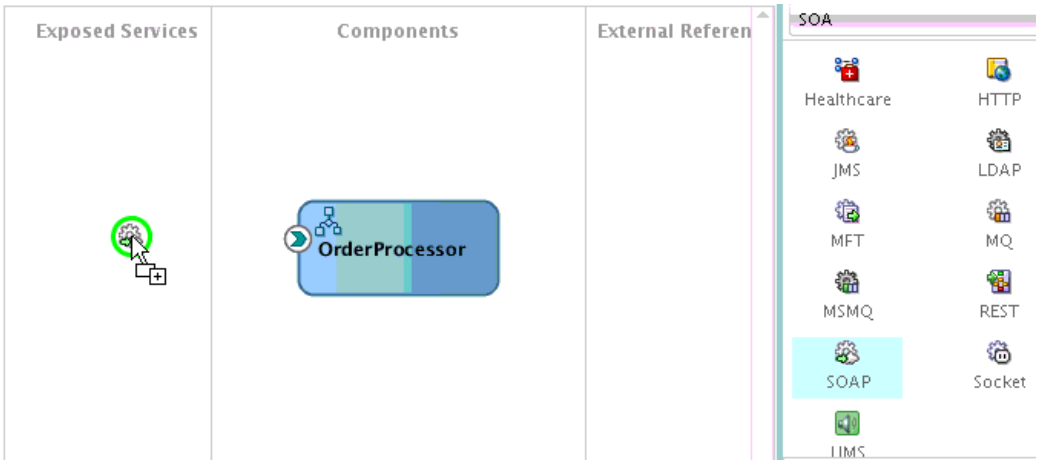
You can use the Components window in the SOA Composite Editor to drag and drop service binding components to the composite.

To add a service binding component:

1. In the Components window, drag a **SOAP** web service to the *left* **Exposed Services** swimlane to define the service interface.

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

Figure 2-5 Adding a SOAP Web Service to a Composite



A specific dialog for the selected service is displayed. Table 2-7 describes the available editors.

Table 2-7 Service Editors

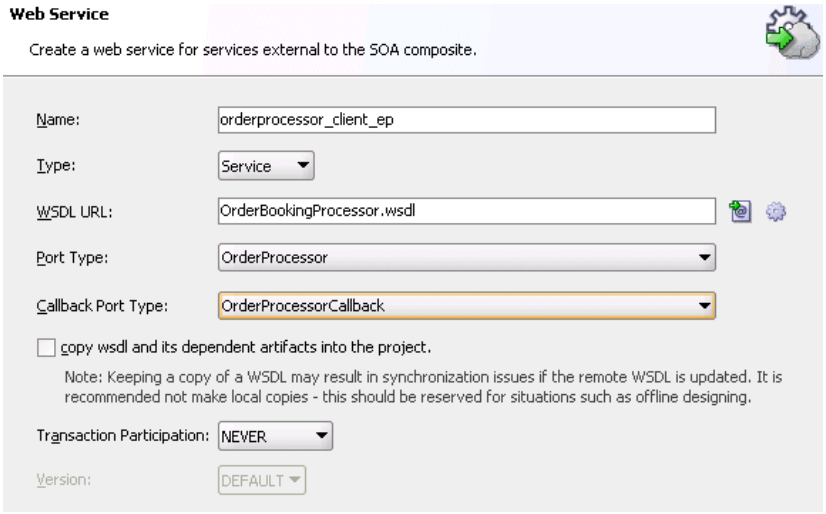
Dragging This Service...	Invokes The...
SOAP	Create Web Service dialog to create a web invocation service.
Adapters	Adapter Configuration Wizard to guide you through integration of the service with database tables, database queues, file systems, FTP servers, Java Message Services (JMS), IBM WebSphere MQ, Oracle User Messaging Service, Oracle BAM 11*g* servers, LDAP server, Coherence cache, sockets, cloud adapters, or Oracle E-Business Suite, JDE World, or SAP applications.
ADF-BC	Create ADF-BC Service dialog to create a service data object (SDO) invocation service.
B2B	B2B Configuration Wizard to guide you through selection of a document definition.
Healthcare	Healthcare Configuration Wizard to guide you through integration with a healthcare system.
EJB	Create EJB Service to create an Enterprise JavaBeans service for using SDO parameters or Java interfaces with Enterprise JavaBeans.
HTTP	Create HTTP Binding Wizard to create HTTP binding. This wizard enables you to invoke SOA composite applications through HTTP POST and GET operations.
Direct	Create Direct Binding Service dialog to invoke a SOA composite application and exchange messages over a remote method invocation (RMI) in the inbound direction.
REST	Create REST Binding dialog to integrate REST operations as service or reference binding components.
MFT	MFT Configuration Wizard to create an MFT source or target.

2. Configure the settings for the service. For help with a service editor, click **Help** or press **F1**. When you add a web service, you must select the WSDL file to use. For information, see How to Define the Interface (WSDL) for a Web Service.

3. 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.

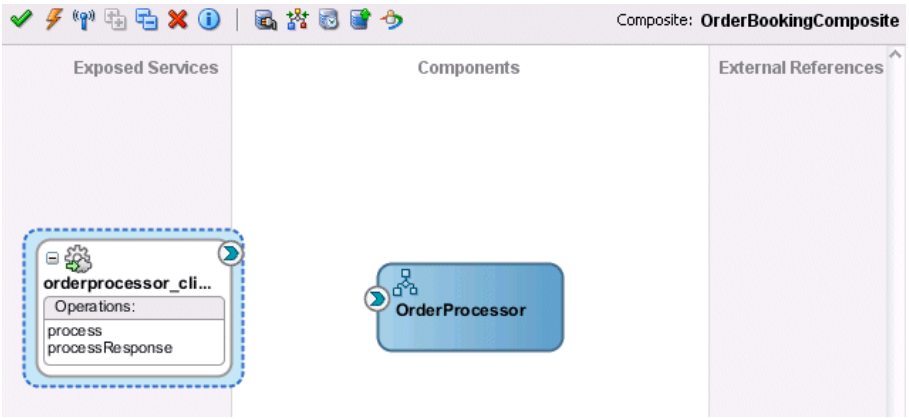
Figure 2-6 Create Web Service Dialog



4. 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_name*** file (for this example, named **OrderBookingComposite**).

Figure 2-7 Web Service in Composite



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

How to Define the Interface (WSDL) for a Web Service

As described in 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 define the interface (WSDL) file for the web service.

To define the interface (WSDL) for a web service:

1. From the **Technology** section, drag a **SOAP** web service to the *left* **Exposed Services** swimlane.

   This invokes the Create Web Service dialog shown in Figure 2-6.

2. Enter the details shown in Table 2-8:

Table 2-8 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.

3. Select the WSDL file for the service. There are three methods for selection:

   - Defining a New WSDL Using a Schema
   - Selecting an Existing WSDL
   - Automatically Defining a Service Interface WSDL from a Component

4. 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**.

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

6. Note the additional fields described in Table 2-9:

Table 2-9 Create Web Service Dialog Fields and Values

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

7. Click **OK**.

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

Note:

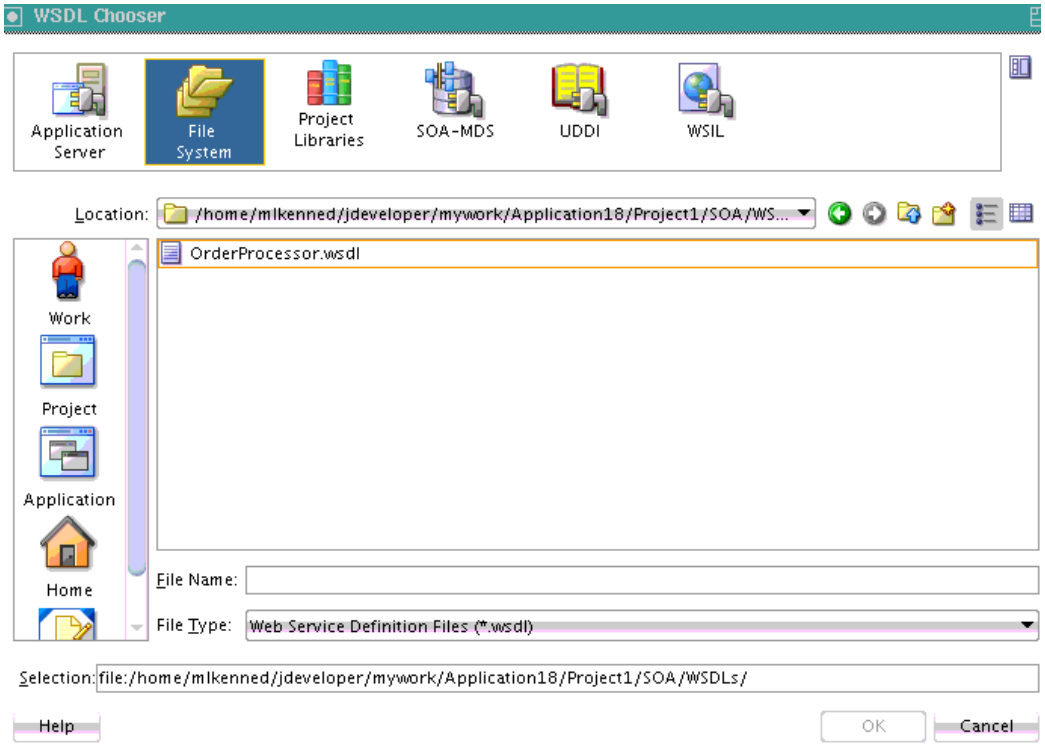
- Do *not* manually update the WSDL location in the WSDL file in **Source View**. This action is not supported. Only updates made in **Design View** are supported.
- WSDL namespaces must be unique. Do not just copy and rename a WSDL. Ensure that you also change the namespaces.

Defining a New WSDL Using a Schema

Define a new WSDL using an existing schema or define a new schema.

1. To the right of the **WSDL URL** field, click the **Find existing WSDLs** (first) icon.
2. At the top, click **File System**.
3. Select an existing WSDL file from the local file system (for this example, **OrderProcessor.wsdl** is selected). Figure 2-8 provides details.

Figure 2-8 WSDL File Selection



Selecting an Existing WSDL

Select a WSDL created when defining a component interface. The WSDL can be selected from the project/application browser.

1. To the right of the **WSDL URL** field, click the **Find existing WSDLs** (first) icon.
2. At the top, click **SOA-MDS**. This action enables you to use existing WSDL files from other applications.

Automatically Defining a Service Interface WSDL from a Component

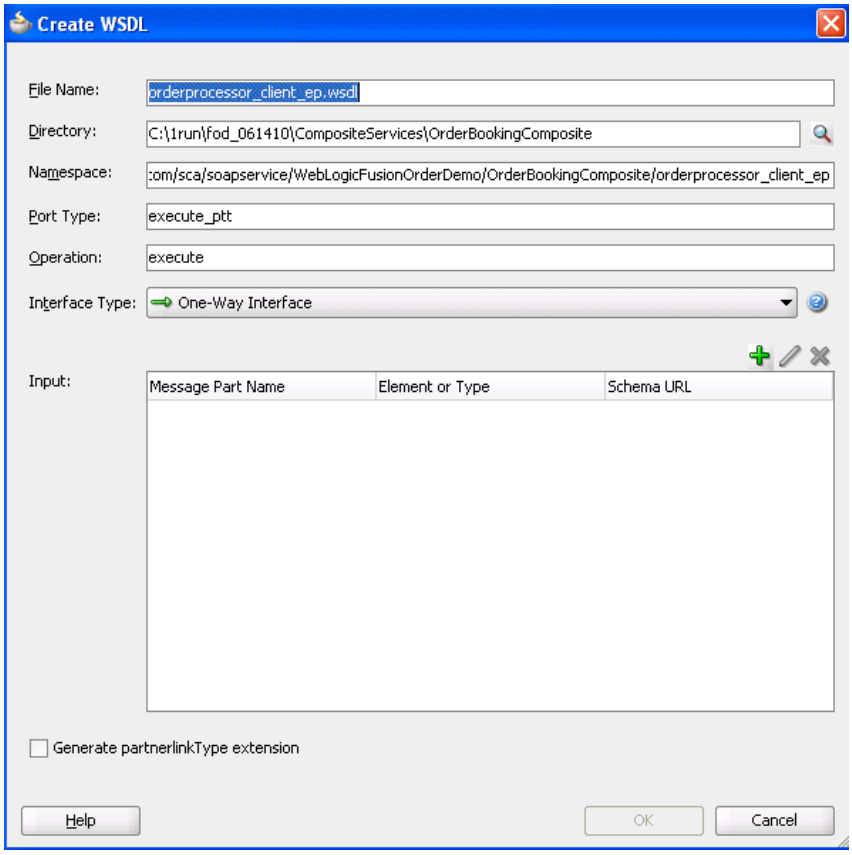
Automatically define a service interface WSDL from a component.

1. To the right of the **WSDL URL** field, click the **Generate WSDL from schemas** (second) icon to automatically generate a WSDL file from a schema.

   Figure 2-9 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 2-9 Automatic Generation of WSDL File



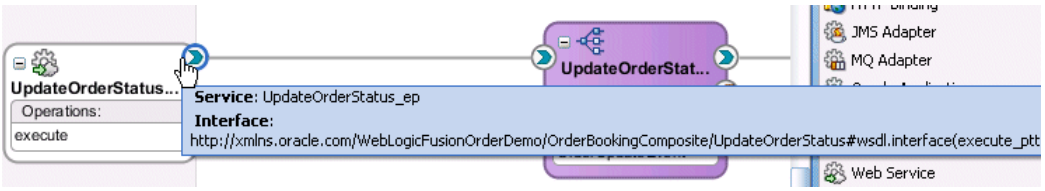
How to View Schemas

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:

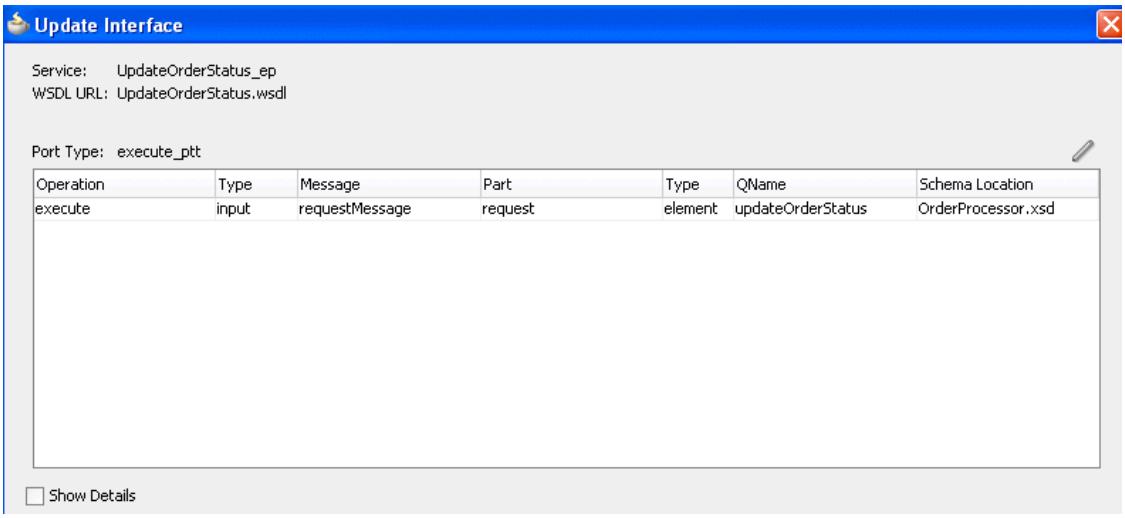
1. Double-click the small arrow handle that appears on the specific binding component or service component. Figure 2-10 provides details.

Figure 2-10 Selection of Inbound Interface Handle



The Update Interface dialog shown in Figure 2-11 displays all schemas currently used by the WSDL file.

Figure 2-11 Update Interface Dialog



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

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 2-10 provides an overview.

Table 2-10 Starting Service Wizards and Dialogs

Double-Click This Service...	To...
SOAP	Display the Update Service dialog.
Adapters	Re-enter the Adapter Configuration Wizard.
ADF-BC	Display the Update Service dialog.
B2B	Re-enter the B2B Configuration Wizard.
Healthcare	Re-enter the Healthcare Configuration Wizard.
EJB Service	Display the Update Service dialog.
HTTP	Re-enter the HTTP Binding Wizard.
Direct	Re-enter the Update Service dialog.

Table 2-10 (Cont.) Starting Service Wizards and Dialogs

Double-Click This Service...	To...
REST	Re-enter the REST Binding dialog.
MFT	Re-enter the MFT Configuration Wizard.

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 are removed.

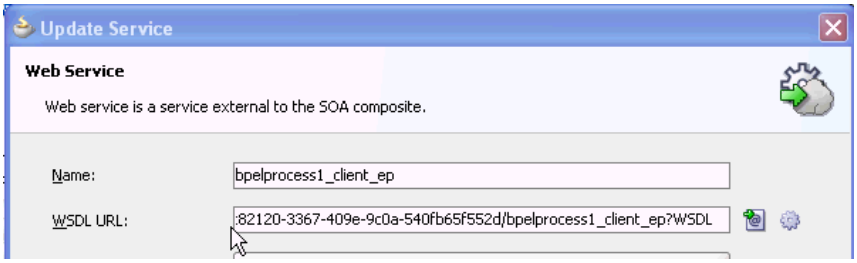
What You May Need to Know About Using the Same Namespace in Different WSDL Files in the Same Composite

Having two different WSDL files with the same fully-qualified namespace in the same SOA composite application is ambiguous and not supported. This causes the application to fail during compilation with duplicate definition errors. Ensure that you use unique namespaces for every WSDL file.

What You May Need to Know About WSDL Browsing in the Resources Window When the SOA Infrastructure Uses Both Internal and External Oracle HTTP Servers

When the SOA Infrastructure is configured in the **Server URL** field of the SOA Infrastructure Common Properties page in Oracle Enterprise Manager Fusion Middleware Control to use both internal and external Oracle HTTP servers, you cannot browse for WSDL URLs using the Resources window. However, you can paste the correct WSDL URL in the **WSDL URL** field of the Update Service dialog for the web service binding component. Figure 2-12 provides details.

Figure 2-12 WSDL URL Field



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.

How to Add a Reference Binding Component

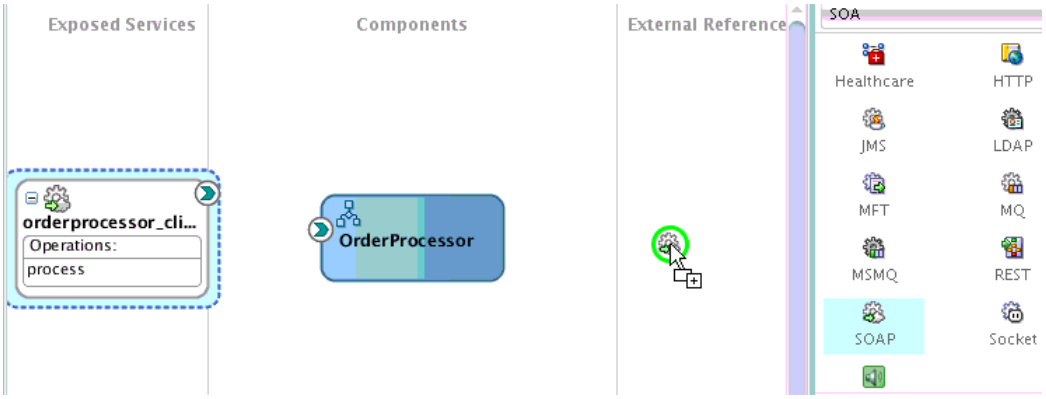
You can use the Components window from the SOA Composite Editor to drag and drop reference binding components into the composite.

To add a reference binding component:

1. From the Components window, select **SOA**.
2. From the **Technology** list, drag a service to the *right* **External References** swimlane.

   Figure 2-13 shows a web service being added to the designer.

Figure 2-13 Adding a SOAP Web Service to the Composite



A specific dialog or wizard for the selected reference displays. Table 2-11 describes the available editors.

Table 2-11 Reference Editors

Dragging This Service...	Invokes The...
SOAP	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, Oracle User Messaging Service, Oracle BAM 11*g* servers, LDAP server, Coherence cache, sockets, cloud adapters, or Oracle E-Business Suite, JDE World, or SAP applications.
ADF-BC	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.

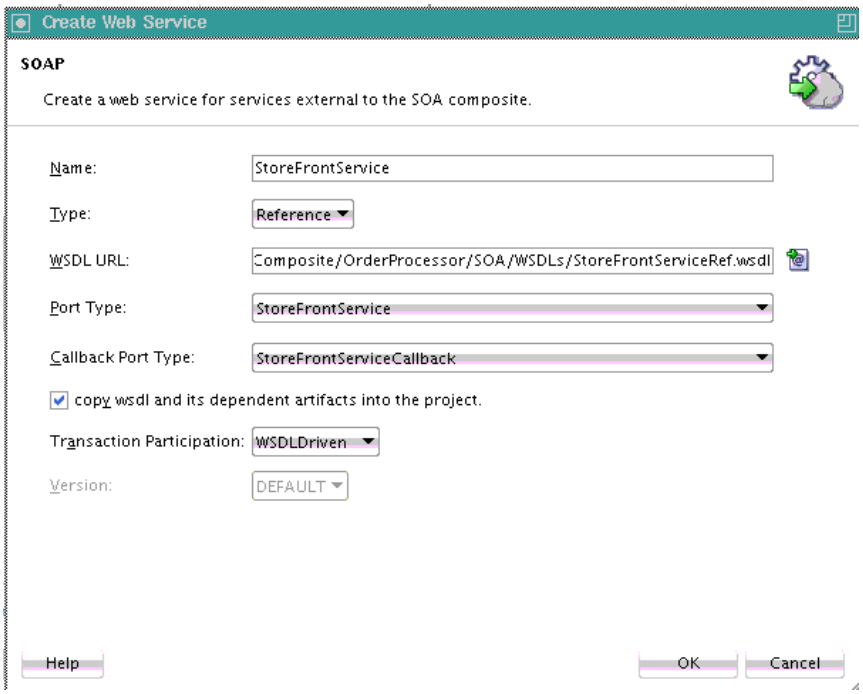
Table 2-11 (Cont.) Reference Editors

Dragging This Service...	Invokes The...
Healthcare	Healthcare Configuration Wizard to guide you through integration with a healthcare system.
EJB	Create EJB Service dialog to create an Enterprise JavaBeans service for using SDO parameters with Enterprise JavaBeans.
HTTP	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	Create Direct Binding Service Dialog to invoke an Oracle Service Bus flow or another SOA composite application.
REST	Create REST Binding dialog to integrate REST operations as service or reference binding components.
MFT	MFT Configuration Wizard to create an MFT source or target.

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.

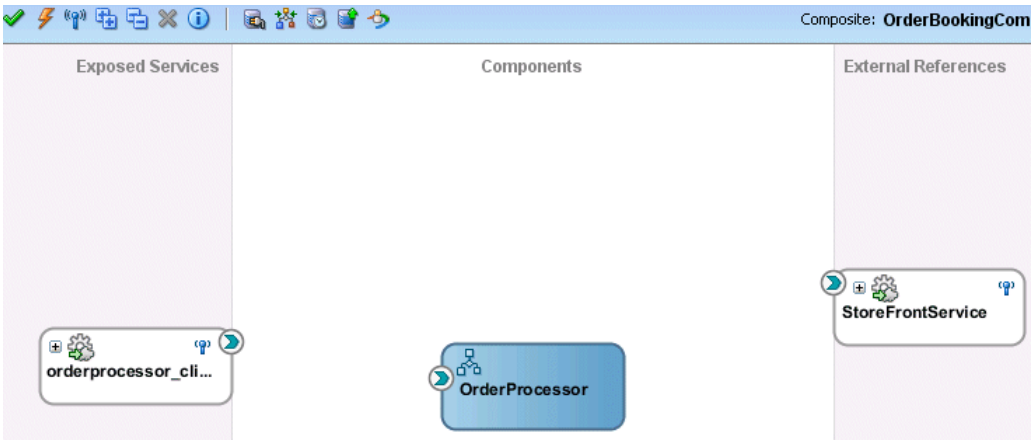
Figure 2-14 Create Web Service Dialog



5. Click **OK**.

Figure 2-15 shows the **StoreFrontService** reference binding component added in the right swimlane of the SOA composite application.

Figure 2-15 SOAP Web Service in the Composite



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

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 is by wiring the service component to the necessary target service component. When a new reference is added, the service component is notified and makes appropriate changes. 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 that 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.

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 in the source view of the ***composite_name*** (**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 in the composite 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). Ensure that you change the following places in this file where a WSDL URL is referenced:

- User interface location - used only in Oracle JDeveloper.
- Import: Used during deployment.
- WSDL location in the reference definition: Used at runtime.

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.

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.

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

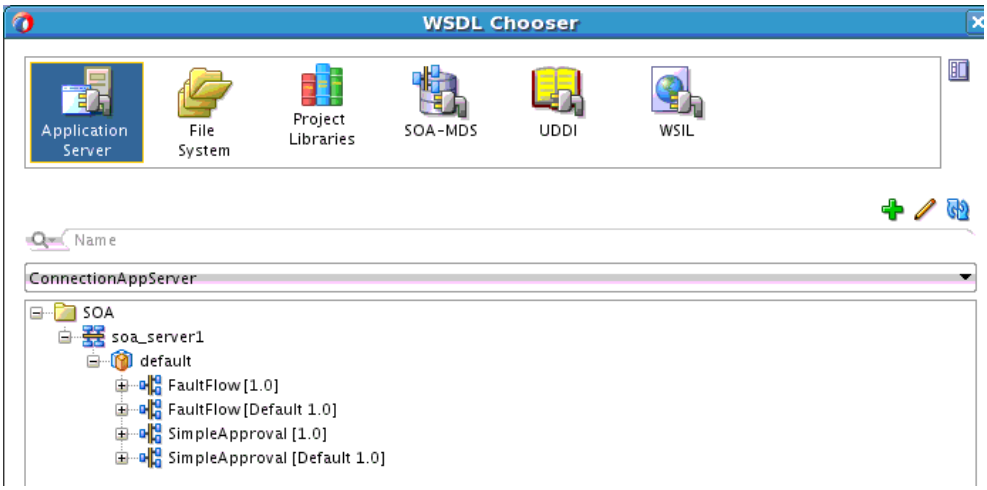
A WSDL URL that does not contain a revision number is processed by the default composite application. This action enables you to always call the default revision of the called service without having to make other changes in the calling composite.

Select the default WSDL to use in the WSDL Chooser dialog in Oracle JDeveloper.

To invoke the default revision of a composite:

1. In the Create Web Service dialog, click the icon to the right of the **WSDL URL** field to invoke the WSDL Chooser dialog.
2. At the top, select **Application Server** or **WSIL**.
3. Expand the nodes to list all deployed composites and revisions. The default revision is identified by the word **Default** in the title (for example, **FaultFlow [Default 1.0]**).

Figure 2-16 WSDL Chooser Dialog



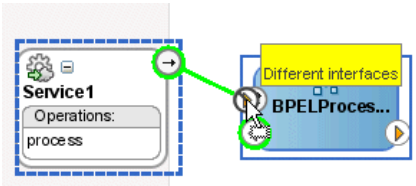
4. Select the appropriate default endpoint and click **OK**.

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.
- The port type and the namespace are used to uniquely identify an interface.
- 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 2-17 provides details.

Figure 2-17 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.

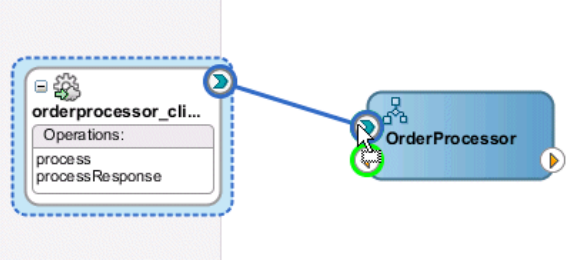
How to Wire a Service and a Service Component

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

To wire a service and a service component:

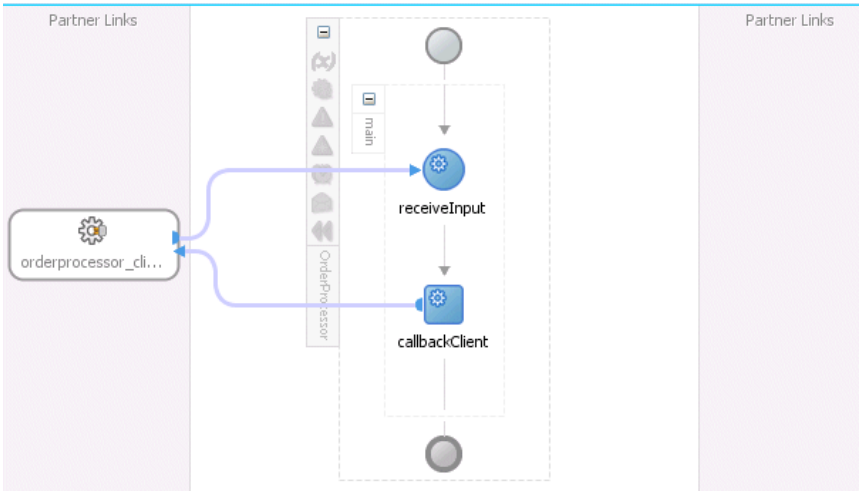
1. From a service reference handle, drag a wire to the service component interface, as shown in Figure 2-18.

Figure 2-18 Wire Connection



2. If the service component is a BPEL process, double-click the BPEL process to open Oracle BPEL Designer. Note that the service displays as a partner link in the left swimlane, as shown in Figure 2-19.

Figure 2-19 Display of the Service as a Partner Link in the BPEL Process



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

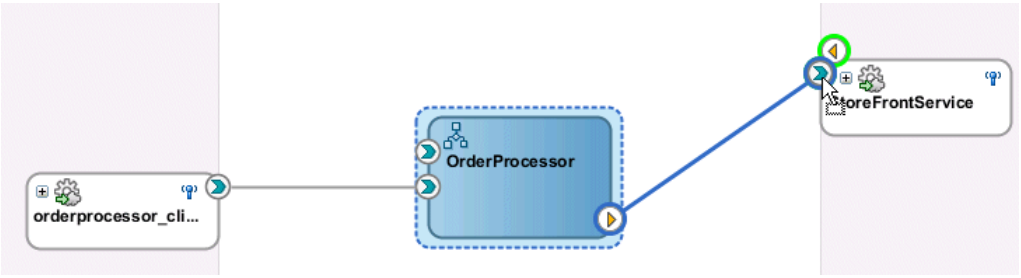
How to Wire a Service Component and a Reference

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

To wire a service component and a reference:

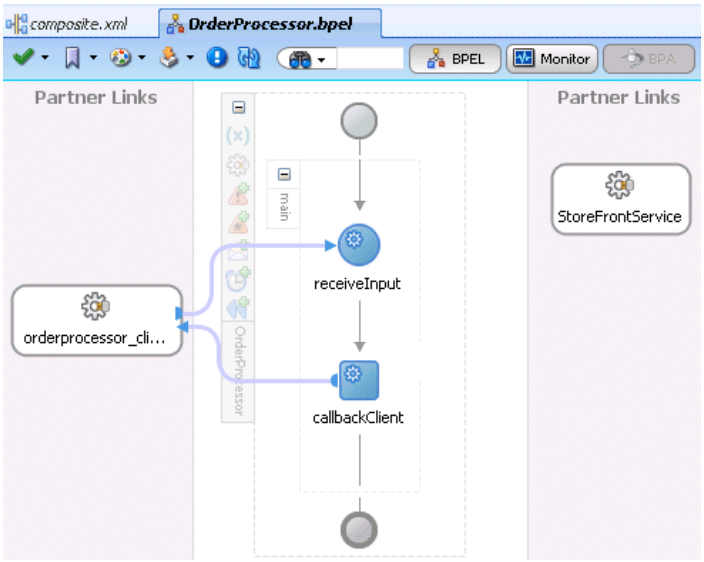
1. In the Applications window, double-click ***composite_name*** or single-click ***composite_name*** above the designer.
2. From the service component, drag a wire to the reference, as shown in Figure 2-20.

Figure 2-20 Wiring of a Service Component and Reference



3. If the service component is a BPEL process, double-click the BPEL process to open Oracle BPEL Designer. Note that the reference displays as a partner link in the right swimlane, as shown in Figure 2-21.

Figure 2-21 Display of the Reference as a Partner Link in the BPEL Process



4. Select **Save All** from the **File** main menu.
5. In the Applications window, select the ***composite_name*** file.
6. Click the **Source** tab to review what you have created.

   The `orderprocessor_client_ep` service binding component provides the entry point to the composite.

```
<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 appears.

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

A reference binding component named `StoreFrontService` appears. The reference provides access to the external service in the outside world.

```
<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>
```

The communication (or wiring) between service components is as follows:

- 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.

```
<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>
```

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 service 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 .

  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 , 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 How to View Schemas for details about the Update Interface dialog.

Adding Descriptions to SOA Composite Applications

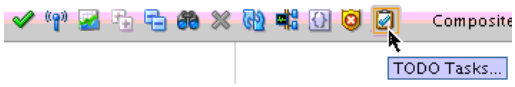
You can add a description of the SOA composite application that is displayed when you place your cursor over the **TODO Tasks** icon above the composite. The description can describe the actions of the services, references, and service components in the SOA composite application.

How to Add Descriptions to SOA Composite Applications

To add descriptions to SOA composite applications:

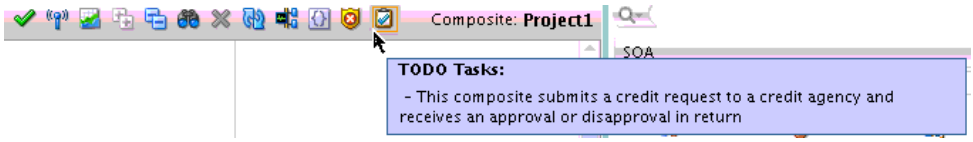
1. Above the SOA Composite Editor, click the **TODO Tasks** icon. Figure 2-22 provides details.

Figure 2-22 To Do Tasks Icon



2. Double-click in the table row, and add the description.
3. When complete, click outside the table row, then click **Close**.
4. Place the cursor over the **TODO Tasks** icon above the SOA composite application to display the description. Figure 2-23 provides details.

Figure 2-23 Description of SOA Composite Application



Renaming, Deleting, and Moving Components and Artifacts

You can rename, delete, and move some components (also known as refactoring) and artifacts in the following sections of Oracle JDeveloper.

- SOA Composite Editor

  Enables you to rename and delete components. These actions impact Oracle SOA Suite metadata (and not necessarily specific artifacts).

- Applications window

  Enables you to rename, delete, and move artifacts such as WSDLs, schemas, and so on. These actions impact Oracle JDeveloper artifacts.

Note:

Do not perform refactoring tasks with Oracle BPEL Designer, Human Task Editor, and other editors open. If you do, ensure that you then close and reopen the editors after refactoring. For example, assume you have a BPEL process open, then rename the BPEL process WSDL file in the Applications window. This changes the underlying BPEL file, but Oracle BPEL Designer does not reflect this change and becomes unsynchronized unless you completely exit it. Close and then reopen Oracle BPEL Designer. The changes are then synchronized.

How to Rename and Delete Components in the SOA Composite Editor

Table 2-12 describes the refactoring tasks that you can perform in the SOA Composite Editor, along with known limitations. Carefully review these restrictions before using this feature.

Table 2-12 Refactoring Components

Action	SOA Composite Editor Steps
Rename a service component or binding component	1. Right-click a component and select **Rename**. Once renamed, all references to the component in the composite are updated. Note the following restrictions: • You cannot rename human workflow, subprocess, or business rule components.
Delete a service component, binding component, or BPEL subprocess	1. Right-click a component or subprocess and select **Delete**.
Move a service component or binding component to another folder	You cannot perform this task from the SOA Composite Editor.

How to Rename, Move, and Delete Artifacts in the Applications Window

Table 2-13 describes the refactoring tasks that you can perform in the Applications window, along with known limitations. Carefully review these restrictions before using this feature.

Table 2-13 Refactoring Component Artifacts

Action	Applications Window Steps
Rename a service component or binding component artifact	1. Right-click a component file, and select **Refactor** > **Rename**. Note the following restrictions: • Component implementation files (**.bpel**, **.mplan**, and so on) are not renamed when the component is renamed in the SOA Composite Editor. This does not cause issues. If you want to rename the implementation files to the same name, use the Applications window. • You cannot rename human workflow, subprocess, or business rule components. • Renaming or moving of business rule and human task artifacts is not supported. For example, you can rename a human task schema file (for example, **HumanTaskPayload.xsd**), but references to this XSD in the **.task** file are not updated. • You cannot rename port types, operations, and elements in the WSDL and XSD editors. • Do not rename a directory or artifact with blank spaces. Spaces in names lead to invalid references. • You can rename SOA projects and composites.

Table 2-13 (Cont.) Refactoring Component Artifacts

Action	Applications Window Steps
Delete a service component, binding component, or BPEL subprocess artifact	1. Right-click a component file, and select **Refactor** > **Delete**. Note the following restrictions: • When you delete an artifact in the Applications window, you are prompted with a message that includes a **Show Usages** option. When **Show Usages** is selected, any usages or references to the artifact from within files are displayed. When the **Delete** option is executed, only the subprocess file is deleted and *no* references are removed. Ensure that you first select **Show Usages** and manually remove references to the file to delete.
Move a service component or binding component to another folder	1. Right-click a component file, and select **Refactor** > **Move**. Note the following restrictions: • Moving a database adapter artifact causes problems because the database adapter has many artifacts that are implicitly referenced by name and must be in the same directory. • You cannot move component implementation files (**.mplan**, **.bpel**, .**sbpel**, **.task**, **.rules**, .**spring**, and so on) in the Applications window. However, these files can be renamed. • Do not move a directory or artifact name with blank spaces. Spaces in names lead to invalid references. • If you move an XSLT file, you lose capabilities such as the current expansion/scrolled state and which item was last selected in the XSLT Map Editor. This is because a **NonDeployedFiles** directory is created in the same folder as the XSLT file. This folder is the default place for test files, dictionary files, report files, DVM/XREF test support files, and so on. This directory is not moved if an XSLT file is moved because the folder contains files used for multiple XSLT files and there is no direct connection between the XSLT file and the file names that may be in the folder.

Viewing Component Details in the Property Inspector

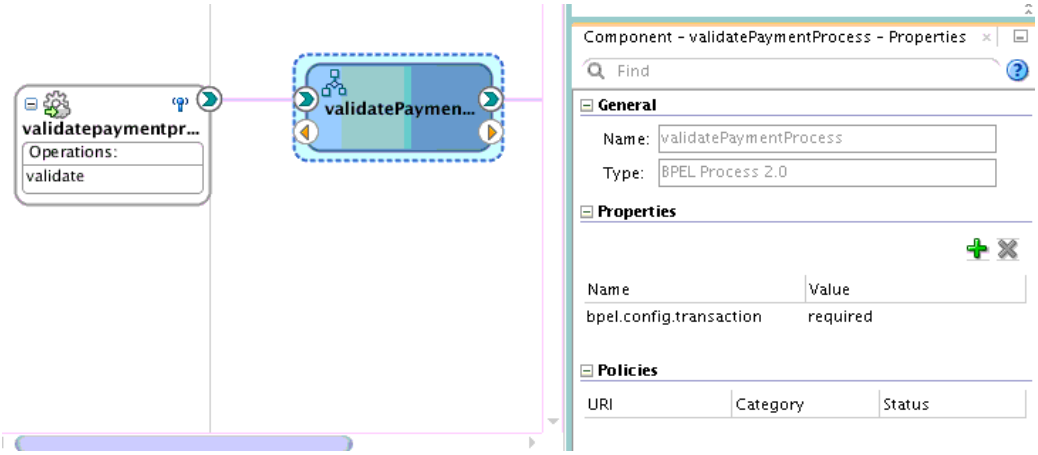
The Property Inspector displays details about the selected service component or binding component in the SOA Composite Editor.

To view properties in the Property Inspector:

1. Select a service, service component, or reference. For this example, a BPEL process service component is selected.

   The Property Inspector is refreshed to display general component details, a section for adding deployment descriptor properties, and attached security policies. Figure 2-24 provides details.

Figure 2-24 Property Inspector



You can also use the Property Inspector to edit BPEL activities in Oracle BPEL Designer. For more information, see How to Edit BPEL Activities in the Property Inspector. and How to Define Deployment Descriptor Properties in the Property Inspector.

Adding Security Policies

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 Enabling Security with Policies and Message Encryption .

Deploying a SOA Composite Application

Deploying a 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 Deploying SOA Composite Applications .

How to Invoke Deployed SOA Composite Applications

You can invoke deployed SOA composite applications from your SOA composite application.

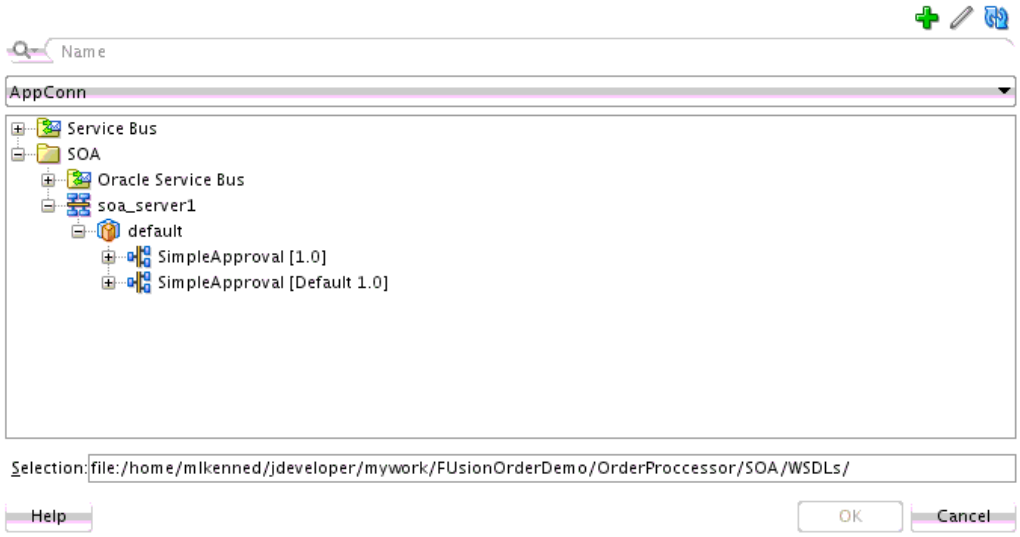
To invoke deployed SOA composite applications:

1. Create a web service or partner link through one of the following methods.
   a. In the SOA Composite Editor, drag a **SOAP** icon from the Components window to the **External References** swimlane.
   b. In Oracle BPEL Designer, drag a **Partner Link** from the **BPEL Constructs** section of the Components window to the *right* swimlane.
2. Access the SOA Resource Browser dialog based on the type of service you created.
   a. From the Create Web Service dialog, click the **Find existing WSDLs** icon. The **Application Server** section of the WSDL Chooser dialog is displayed.

b. From the Edit Partner Link dialog, click the **SOA Resource Browser** icon. The **Application Server** section of the WSDL Chooser dialog is displayed.

3. Select **Application Server** if it is not selected.
4. Expand the tree to display the application server connection to the server on which the SOA composite application is deployed.
5. Expand the application server connection.
6. Expand the **SOA** folder and partition. Figure 2-25 provides details.

Figure 2-25 Browse for a SOA Composite Application



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

For information about creating an application server connection, see Creating an Application Server Connection.

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.

How to Manage Deployed SOA Composite Applications in Oracle JDeveloper

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.

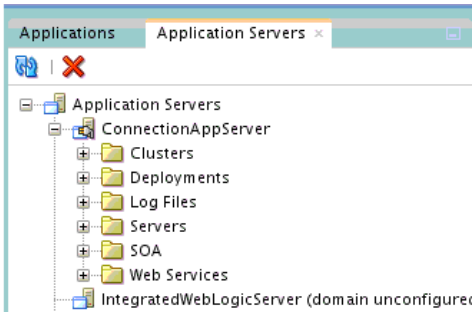
Note:

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** > **Application** > **Connections** > **Application Server Connection** to create a connection.

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

   The **SOA** folder appears, as shown in Figure 2-26. 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 2-26 provides details.

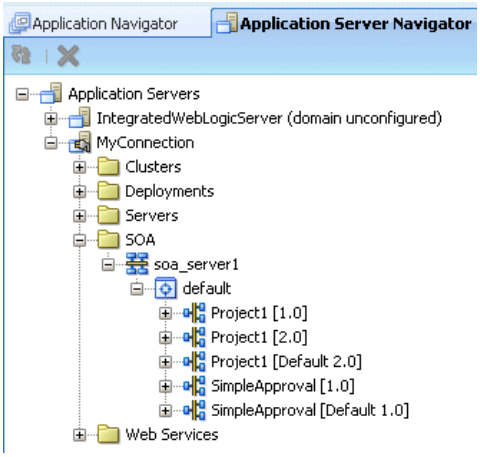
Figure 2-26 Application Server Navigator



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 2-27.

Figure 2-27 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 2-14 provides details.

Table 2-14 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.
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 aborted 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.

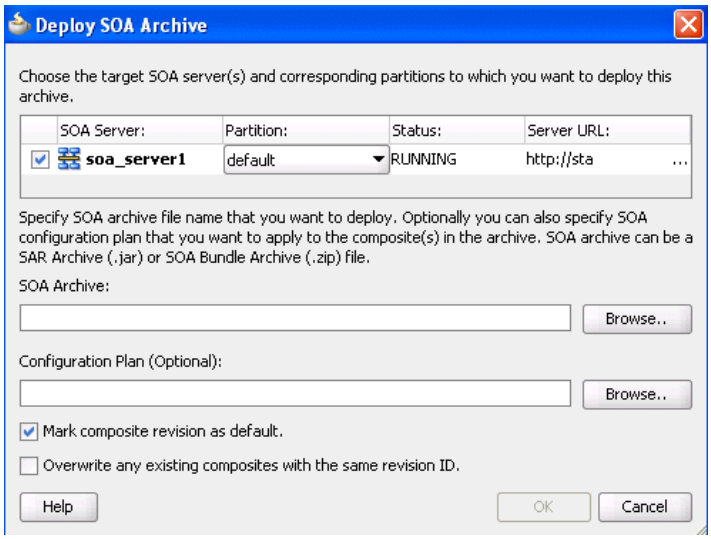
7. If you want to deploy a *prebuilt* SOA composite application archive that includes a deployment profile, right-click the **SOA** folder and select **Deploy SOA Archive**. The archive consists of a JAR file of a single application or a SOA bundle ZIP file containing multiple applications.

   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 2-28 provides details.

Figure 2-28 Deploy SOA Archive Dialog



For more information, see the following documentation:

- Deploying SOA Composite Applications for details about creating a deployment profile and a configuration plan and deploying an existing SOA archive
- *Administering Oracle SOA Suite and Oracle Business Process Management Suite* for details about managing deployed SOA composite applications from Oracle Enterprise Manager Fusion Middleware Control.

How to Test and Debug a Deployed SOA Composite Application

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 *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

In addition to creating a test instance, you can also perform the following testing and debugging tasks in Oracle JDeveloper:

- 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 Automating Testing of SOA Composite Applications.
- Test and debug SOA composite applications with the SOA debugger in Oracle JDeveloper. The SOA debugger reduces the development cycle for a SOA composite application by providing a troubleshooting environment within Oracle JDeveloper. This eliminates the lengthy process of building a SOA composite application in Oracle JDeveloper, deploying it to the SOA Infrastructure, starting Oracle Enterprise Manager Fusion Middleware Control to test or view audit trails and flow traces, and then returning to Oracle JDeveloper to repeat the exercise. For more information, see Debugging and Auditing SOA Composite Applications .

3

Managing Shared Data with the Design-Time MDS Repository

This chapter describes how to manage shared data with the SOA Design-Time Oracle Metadata Services Repository (MDS Repository), including how to create and delete folders, export and import the contents of the `/apps` folder to and from a JAR file, transfer the `/apps` folder contents to another SOA Design-Time MDS Repository, export a Release 11*g* MDS Repository to a JAR file, and use the SOA-MDS Transfer wizard to share data with the SOA Design-Time MDS Repository.

This chapter includes the following sections:

Introduction to SOA Design-Time MDS Repository Management

A file-based, SOA Design-Time MDS Repository is automatically created when you create a SOA composite application. You cannot modify the MDS Repository name, but you can modify it to point to an existing, file-based repository. You typically point it to the version control system (MDS) location. Sharing operations are done against the design-time repository. You cannot perform these operations against a database-backed MDS Repository.

You can perform the following operations against the SOA Design-Time MDS Repository in Oracle JDeveloper:

- Browse the following folder recognized by Oracle SOA Suite in the SOA Design-Time MDS Repository:
  - `/apps`: Contains shared data, including Oracle Service Bus artifacts.
- Create folders directly under the `/apps` folder or a subfolder of `/apps`.

- Delete files and subfolders under the `/apps` folder. The `/apps` folder itself cannot be deleted.
- Export selected contents of the `/apps` folder to a JAR file. The `/apps` folder itself is not included in the JAR file.
- Import the contents of a JAR file under the `/apps` folder. If the JAR file includes `/apps` as the root folder, it is created below the `/apps` folder of the design-time MDS Repository, which gives you a top-level directory structure of `/apps/apps`.
- Transfer the contents of the `/apps` folder of one MDS Repository to another MDS Repository.
- Export an existing MDS Repository (for example, a Release 11*g* database-based MDS Repository) to a JAR file. This JAR file can then be imported into the Release 12*c* design-time MDS Repository.

Introduction to the Default SOA Design-Time MDS Repository Connection

A file-based, SOA Design-Time MDS Repository connection named SOA_DesignTimeRepository is automatically included when you create a SOA composite application. The default directory location is `$JDEV_USER_DIR/soamds`.

This connection provides the following capabilities:

- A file-based MDS Repository for use during design time. A database-based design-time MDS Repository is *not* supported.
- Any MDS Repository can be browsed.
- The default repository location can be modified to point to another folder or version control location.
- All SOA-MDS operations use this SOA Design-Time MDS Repository.
- A wizard enables you to share design-time artifacts from your SOA project with this MDS Repository, such as WSDL and schema files.

Note:

- If you add shared data into the SOA Design-Time MDS Repository, and the repository is backed by a version control system, Oracle SOA Suite does not provide any operations to add this data to the version control system. You must add this shared data to the version control system.
- If you have a Release 11*g* SOA composite application with a preconfigured SOA-MDS repository (`/apps` namespace) in the `adf-config.xml` file, all sharing and consumption operations are performed against the existing repository defined in `adf-config.xml`.

Changing the Default SOA-MDS Location

When you create a SOA composite application, the default SOA-MDS connection named SOA_DesignTimeRepository is automatically included. The `/apps` folder in the SOA design-time MDS Repository is automatically created.

Note:

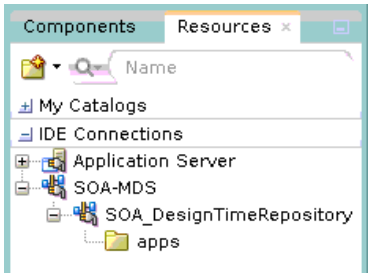
When files from an Oracle JDeveloper project are shared using the SOA_DesignTimeRepository, the original files are moved from the SOA project to the default SOA-MDS repository.

How to Change the Default SOA-MDS Location

To change the default SOA-MDS location:

1. Create a SOA composite application.
2. From the **Window** main menu, select **Resources**.
3. In the Components window, click **Resources**.
4. Expand **SOA-MDS**. The artifacts shown in Figure 3-1 are displayed.
   - The SOA-MDS connection named **SOA_DesignTimeRepository** that was automatically created during SOA composite application.
   - The **/apps** folder in the MDS Repository. This folder is initially empty.

Figure 3-1 Resources Window in Oracle JDeveloper



5. Right-click the **SOA_DesignTimeRepository** connection and select **Properties** to point it to your version control location.

   The Edit MDS-SOA Connection dialog is displayed.

6. In the **MDS Root Folder** field, click **Browse**.
7. Select the version control location for the **/apps** folder, and click **Select**. The SOA-MDS browser only displays the **/apps** and **/soa** folders. Therefore, if **/apps** is not present in the selected version control location, then it is not rendered by the browser.

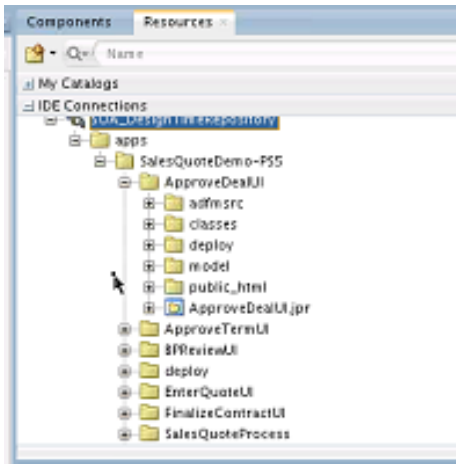
   The specified location is displayed in the Edit MDS-SOA Connection dialog, as shown in Figure 3-2.

Figure 3-2 Edit SOA-MDS Connection Dialog

8. Click **OK**, and expand the **SOA_DesignTimeRepository** connection.

   The **/apps** folder is populated with the location specified in Step 7, as shown in Figure 3-3.

Figure 3-3 Populated /apps folder



Sharing Data with the SOA Design-Time MDS Repository

The SOA-MDS Transfer wizard enables you to share WSDL, XSD, WADL, and XQuery files with the SOA design-time MDS Repository. These files can then be shared with other SOA composite applications.

The wizard first attempts to share files with any existing design-time MDS Repository defined in the current application's `adf-config.xml` file. If no MDS Repository is defined in the `adf-config.xml` file, then artifacts are shared using **SOA_DesignTimeRepository**.

Note:

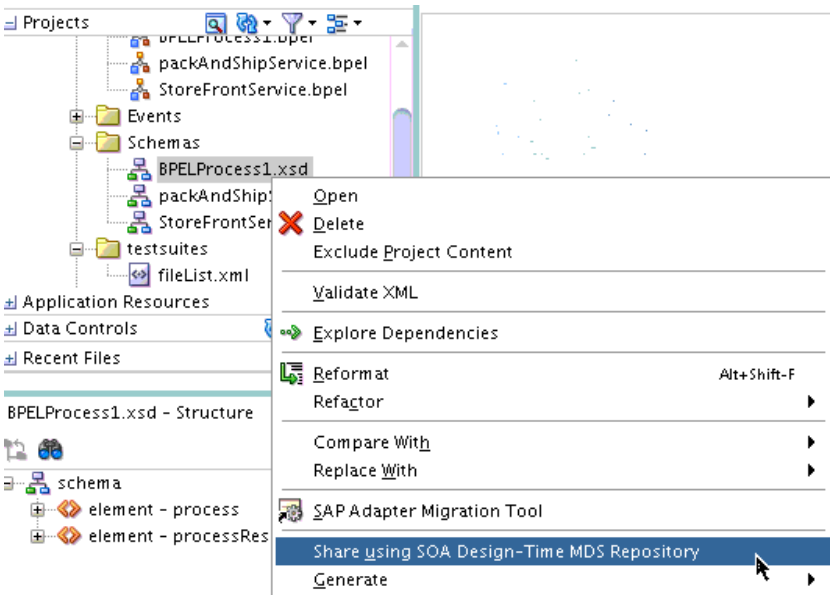
- You can only share XSD, WSDL, WADL, and XQuery files. In addition, only these file types can be transferred from a design-time MDS Repository to a runtime MDS Repository.
- If you right-click an XSD file in the Applications window that was created with the Native Format Builder wizard, the **Share using SOA Design-Time MDS Repository** option is not available.

How to Share Data with the SOA Design-Time MDS Repository

To share data with the SOA design-time MDS Repository:

1. In the Applications window, right-click the file to share (for this example, an XSD file) and select **Share using SOA Design-Time MDS Repository**. Figure 3-4 provides details.

Figure 3-4 Data Sharing with the SOA Design-Time MDS Repository



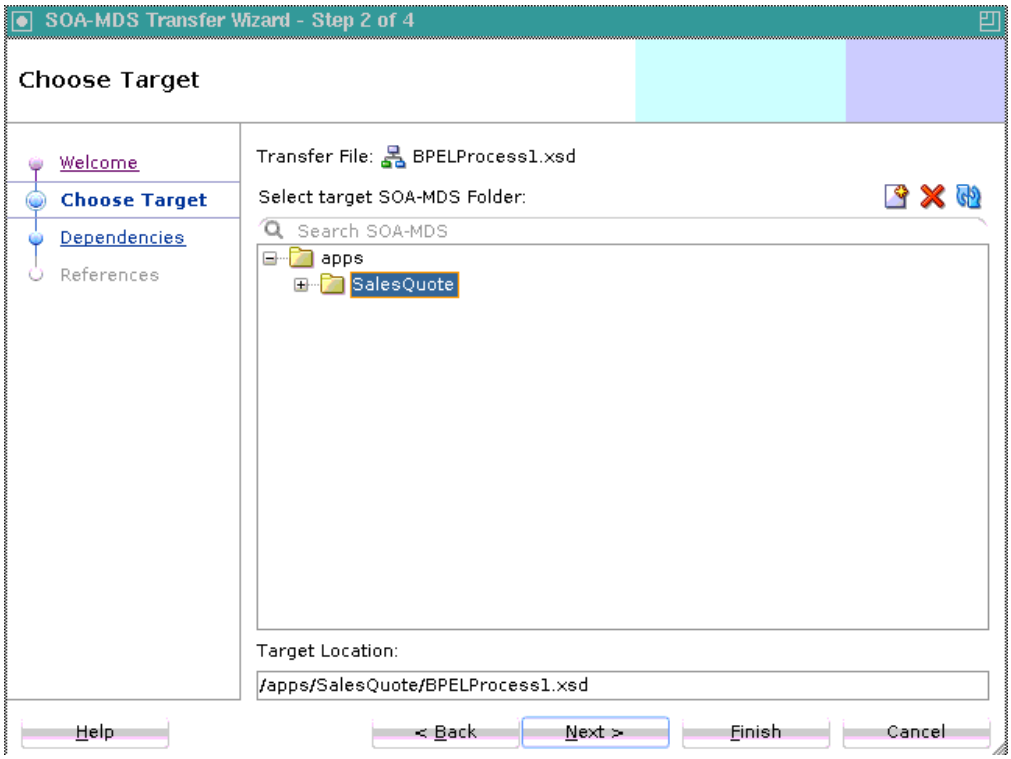
The SOA-MDS Transfer wizard - Welcome page is displayed and indicates that the file you selected is to be transferred to the SOA design-time MDS Repository.

2. Click **Next**.

   The Choose Target dialog is displayed.

3. Browse the design-time MDS Repository and select the target folder in which to share the selected artifact, and click **Next**. You can also create a subfolder in which to share the file or search for an existing folder. Figure 3-5 provides details.

Figure 3-5 SOA-MDS Transfer Wizard - Choose Target Page

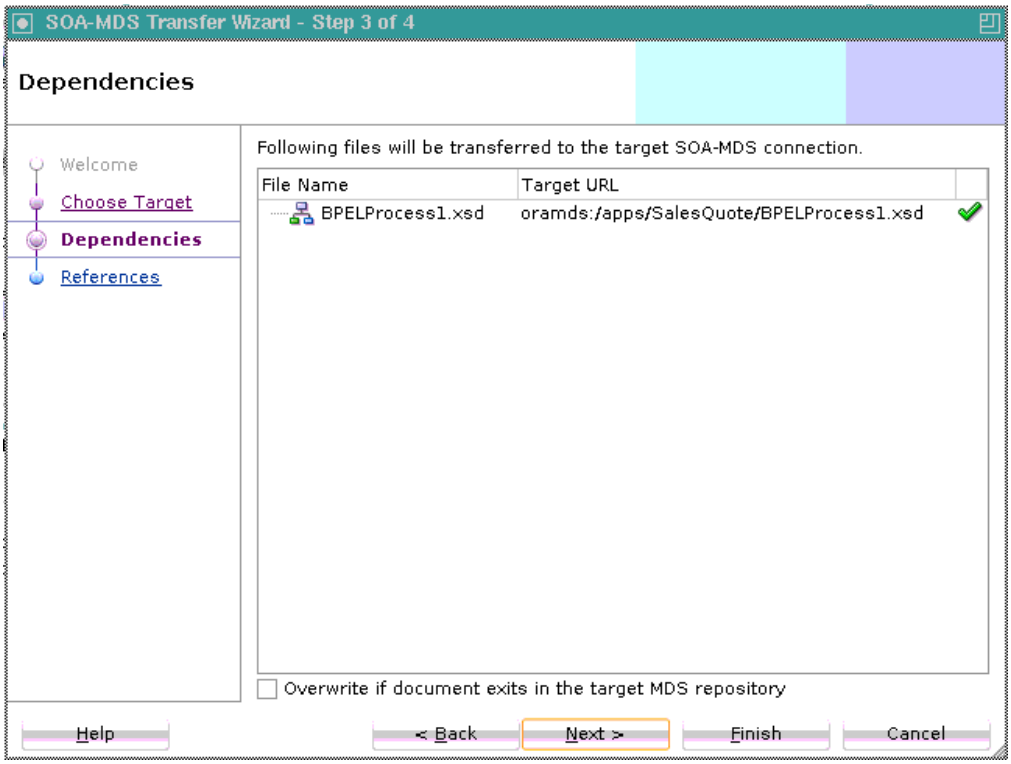


   The Dependencies dialog is displayed.

4. Review the files to transfer to the target **oramds** URL location in the design-time MDS Repository, as shown in Figure 3-6.

   Additional dependent files can also be displayed. For example, assume you select a WSDL file. Because the WSDL file can have dependencies on schema files (potentially more than one file), those XSD files are also displayed.

Figure 3-6 SOA-MDS Transfer Wizard - Dependencies Page



The green checkmark indicates that the file path is correct and resolvable.

Note:

- If the URL is not accessible, an error icon is displayed. For example, assume you are transferring a WSDL file that has dependencies on schemas that traverse several parent levels (for example, **../../../**). If such references are present in the WSDL and you do not select the correct target folder, the URL may go beyond the **/apps** folder, which is not accessible to the SOA Infrastructure. The error icon indicates the target URL is not accessible, and you cannot proceed with the transfer. You must cancel or click **Back** to select a different target folder. In summary, the destination for all URLs must begin with the **/apps** folder.
- File transfers are in relation to the **/apps** folder in the target SOA design-time MDS Repository. Dependent files are typically at the same parallel level. For example, the WSDL file selected for transfer is located in the **WSDLs** folder and the dependent XSD file is located in the **Schemas** folder. Both folders are at the same parallel level under the **SOA** folder of the SOA composite application in the Applications window. However, if the dependent files are at different levels (higher levels than the file that is being shared), you must determine the relative hierarchy of the files. For example, If **foo.wsdl** refers to an XSD file in the location **../../../.xsd**, you must manually create three subfolders under **apps** in the target design-time MDS Repository and share **foo.wsdl** to the lowest folder level so that the XSD can be shared at the **apps** level.

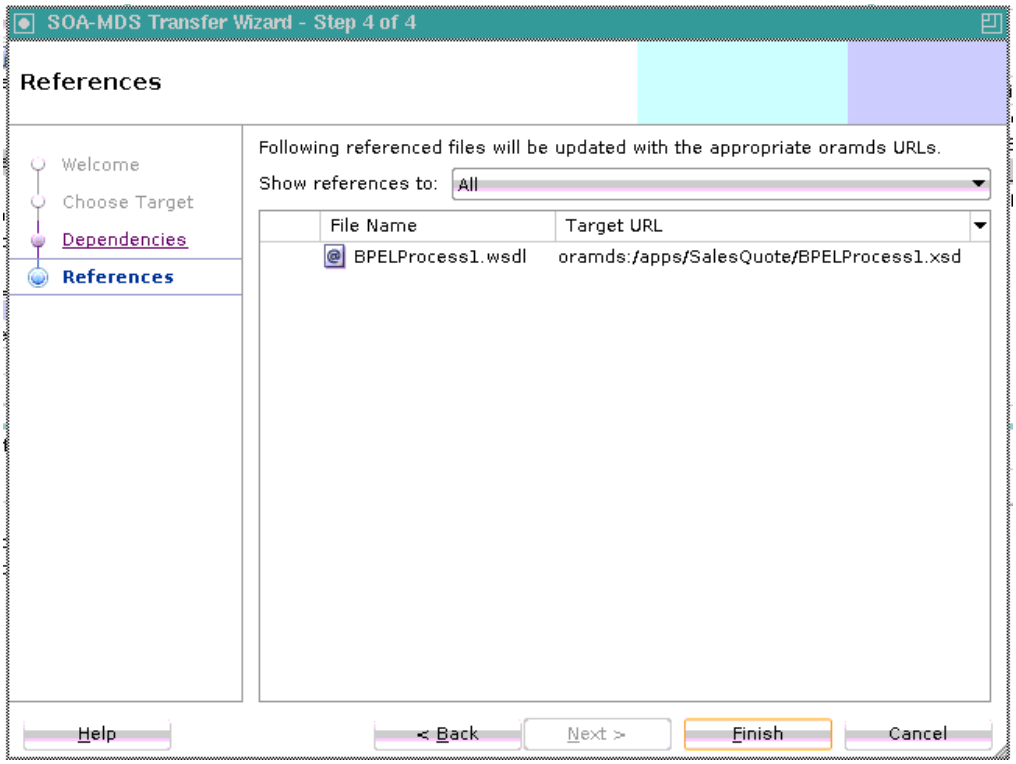
---

5. If you want to overwrite files, select **Overwrite if document exists in the target MDS repository**, then click **Next**. If you do not select this check box, and the files already exist in the target location, no files are transferred and an error message is displayed. You cannot selectively transfer specific files.

   The References dialog is displayed.

6. View the files to be modified after the transfer with the appropriate **oramds** URL, and click **Finish**, as shown in Figure 3-7. This list includes files that are dependent on the files being moved. All dependent files are modified to reflect the **oramds** URL of the file being moved.

Figure 3-7 SOA-MDS Transfer Wizard - References Page



7. Click **OK** when prompted with a message that the transfer completed successfully.

   When complete, the following updates are made:

   - The selected artifacts are displayed beneath the SOA-MDS connection in the Resources window.
   - The **adf-config.xml** file in the Applications window is modified with the `/apps` namespace:

     ```
     <namespace path="/apps" metadata-store-usage="mstore-usage_2"/>
     ```

     The variable that internally points to the SOA design-time MDS Repository home is set:

     ```
     value="${soamds.apps.home}
     ```

   - A reference in the artifact (for example, a WSDL file) is updated to point to the oramds URL location.

Creating and Deleting Subfolders Under the /apps Folder

You can create and delete subfolders under the `/apps` folder in the SOA Design-Time MDS Repository. You cannot delete the `/apps` folder.

How to Create and Delete Subfolders Under the /apps Folder

To create and delete subfolders under the /apps folder:

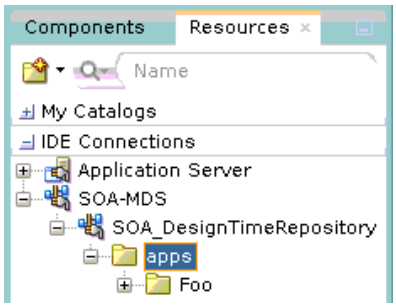
1. Right-click the **/apps** folder or a subfolder of **/apps**, and select **Create Folder** to point it to your version control location.

   The Create Folder dialog is displayed.

2. Enter a name (for this example, `Foo` is entered) for the folder, and click **OK**.

   The folder is created under the **/apps** folder, as shown in Figure 3-8.

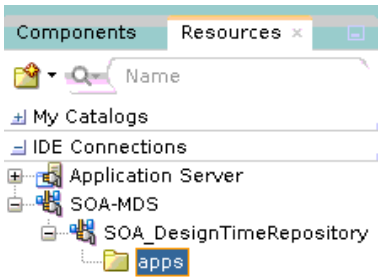
Figure 3-8 New Subfolder Under /apps Folder



3. Right-click the folder to delete (for this example, **Foo**), and select **Delete**.

   The folder is deleted, as shown in Figure 3-9.

Figure 3-9 Subfolder Deleted Under /apps Folder



Exporting the Selected Contents of the /apps Folder to a JAR File

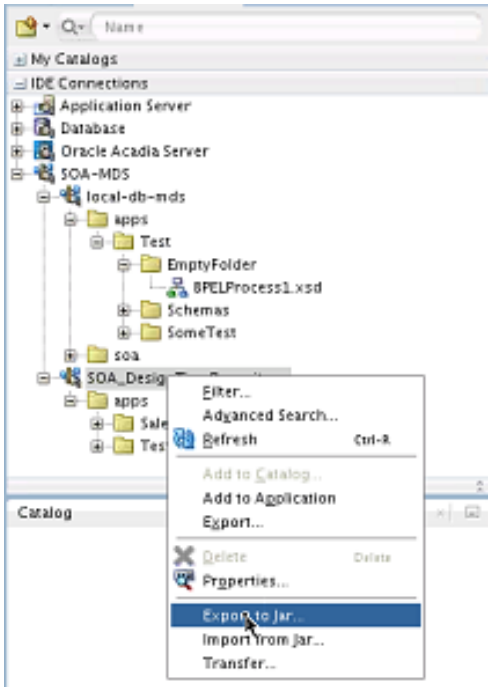
You can export the selected contents of the `/apps` folder in the SOA design-time MDS Repository to a JAR file. The `/apps` folder itself is not exported to the JAR.

How to Export the Selected Contents of the /apps Folder to a JAR File

To export the selected contents of the /apps folder to a JAR file:

1. Right-click the SOA-MDS connection that includes the contents to export (for example, the default **SOA_DesignTimeRepository** connection or another connection), and select **Export to Jar**, as shown in Figure 3-10.

Figure 3-10 Export to Jar Command



The Export to jar dialog is displayed.

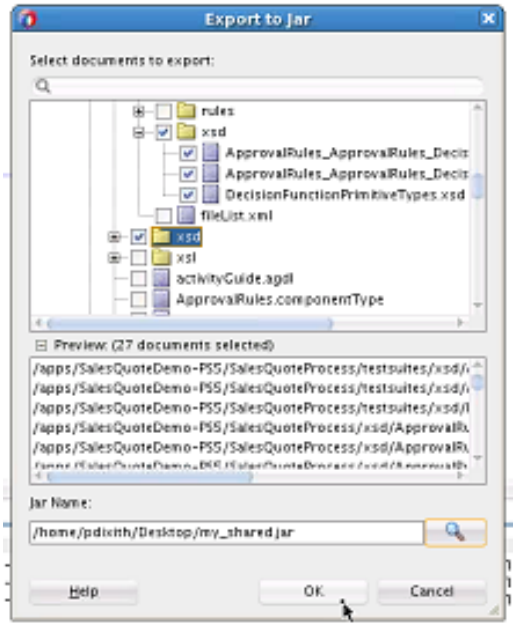
2. Provide values appropriate to your environment, and click **OK**, as described in Table 3-1.

Table 3-1 Export to jar Dialog

Field	Description
Select documents to export	Enter a file or folder name and click **Search** or manually expand the **/apps** folder to identify and select folders and files to export to a JAR file.
Preview Documents Selected	Select to preview the contents to export.
JAR Name	Click **Browse** to select the JAR file to which to export the selected folders and files.

The Export to jar dialog looks as shown in Figure 3-11.

Figure 3-11 Export to jar Dialog



3. Click **OK** when prompted with a message indicating that the export was successful.

Importing the Contents of the JAR File into the /apps Folder

You can import the contents of a JAR file to the `/apps` folder of a SOA design-time or database-backed MDS Repository. If you import a JAR file that includes `/apps` as the root folder, it is created below the `/apps` folder of the design-time MDS Repository, which gives you a top-level directory structure of `/apps/apps`.

How to Import the Contents of the JAR File into the /apps Folder

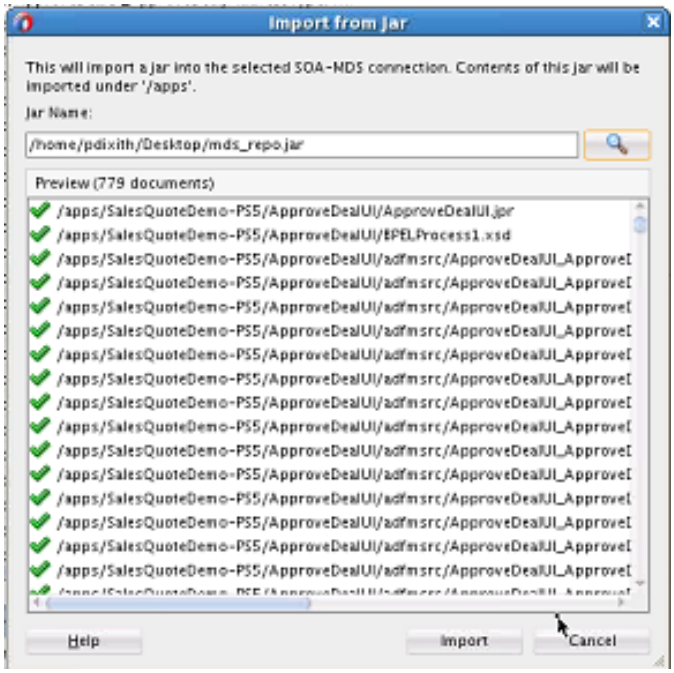
To import the contents of the JAR file into the /apps folder:

1. Right-click the SOA-MDS connection in which to import the JAR file (for example, the default **SOA_DesignTimeRepository** connection or another connection), and select **Import From JAR**.

2. Click **Browse** to select the JAR to import.

   The Import from jar dialog is displayed, as shown in Figure 3-12.

Figure 3-12 Import from jar Dialog

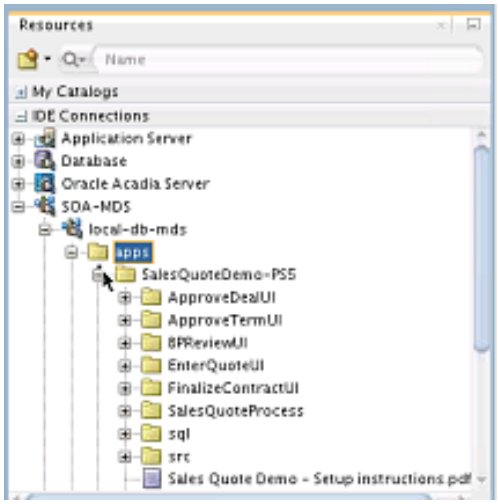


A green checkmark indicates that the contents do not exist in the target repository. If the content exists in the target repository, a warning icon is displayed. You can select to overwrite the content by clicking **Import** or cancel the entire import operation by clicking **Cancel**. You cannot selectively import specific files.

3. Click **Import**. Any artifacts with a warning icon are overwritten.

   The contents of the imported JAR file are displayed under the **/apps** folder, as shown in Figure 3-13.

Figure 3-13 Contents of Imported JAR File in Resources Window



Transferring the Selected Contents of the /apps Folder to Another MDS Repository

You can transfer the selected contents of the `/apps` folder of one MDS Repository to the `/apps` folder of another MDS Repository. There are no limitations on the type of MDS Repository to which to transfer. For example, you can transfer the selected contents of a file-based repository to a database-based MDS Repository, and vice versa.

Note:

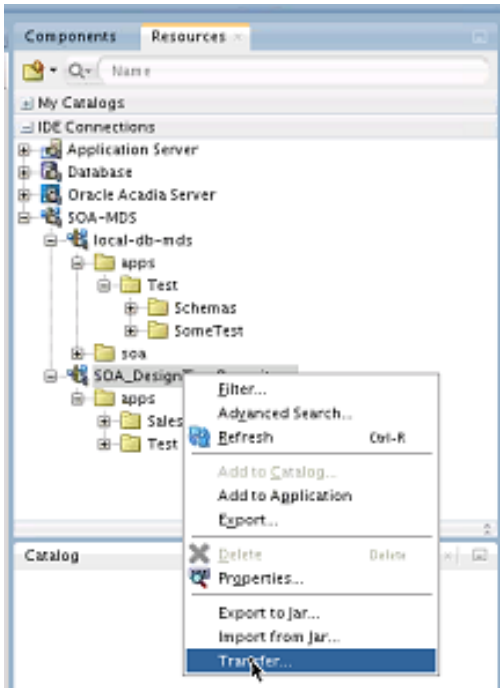
Do not transfer the contents of the `/apps` folder to another MDS Repository with the Oracle BPEL Designer, Human Task Editor, or other editors open. If you do, ensure that you then close and reopen the editors after the transfer completes. An open editor does not reflect the transfer changes and becomes unsynchronized unless you completely exit it.

How to Transfer the Selected Contents of the /apps Folder to Another MDS Repository

To transfer the selected contents of the /apps folder to another MDS Repository:

1. Right-click the SOA-MDS connection that includes the contents to transfer (for example, the default **SOA_DesignTimeRepository** connection or another connection), and select **Transfer**. Figure 3-14 provides details.

Figure 3-14 Transfer Menu Option



The Transfer to SOA-MDS dialog is displayed.

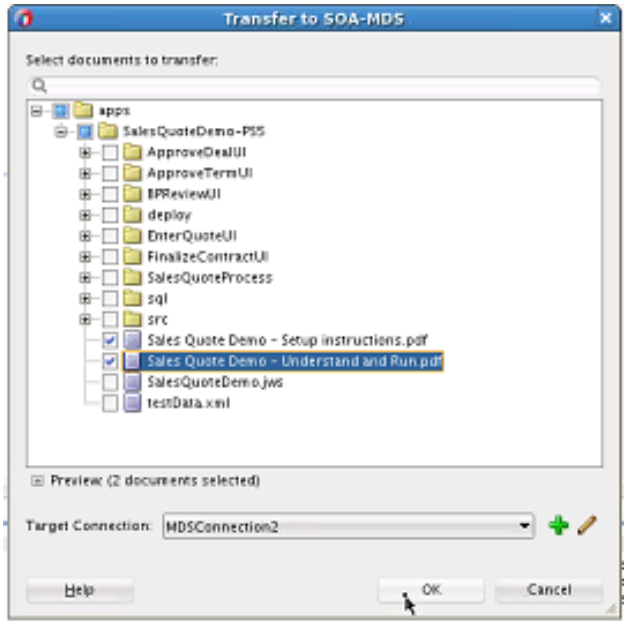
2. Provide values appropriate to your environment, and click **OK**, as described in Table 3-2.

Table 3-2 Transfer to SOA-MDS Dialog

Field	Description
Select Documents to Transfer	Select the contents to transfer.
Preview Documents Selected	Select to preview the contents to transfer.
Target Connection	Select the SOA-MDS connection of the MDS Repository to which to transfer contents.

The Transfer to SOA-MDS dialog looks as shown in Figure 3-15.

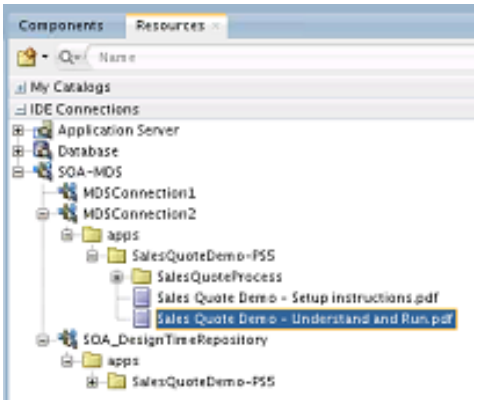
Figure 3-15 Transfer to SOA-MDS Dialog



3. Click **OK** when prompted with a message indicating that the transfer was successful.

   The contents are displayed under the **/apps** folder of the SOA-MDS target connection you selected in the **Target Connection** field in Step 2. Figure 3-16 provides details.

Figure 3-16 Contents Display Under /apps Folder of Selected SOA-MDS Connection



Exporting an Existing Release 11*g* MDS Repository to a JAR File

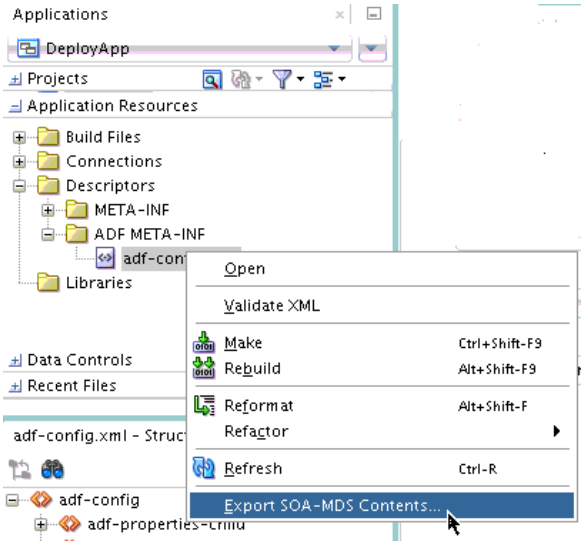
You can export a Release 11*g* MDS Repository to a JAR file that can then be imported into a Release 12*c* design-time MDS Repository. The `adf-config.xml` file is updated with `/apps` and store information. Release 12*c* repositories can also be exported if you have an `adf-config.xml` file with `/apps` defined (meaning you have an existing shared repository).

How to Export an Existing Release 11*g* MDS Repository to a JAR File

To export an existing Release 11*g* MDS Repository to a JAR file:

1. In the Applications window, right-click **adf-config.xml** of the project to export, and select **Export SOA-MDS Contents**. Figure 3-17 provides details.

Figure 3-17 Export of an 11g MDS Repository from the Applications Window



The Export to jar dialog is displayed.

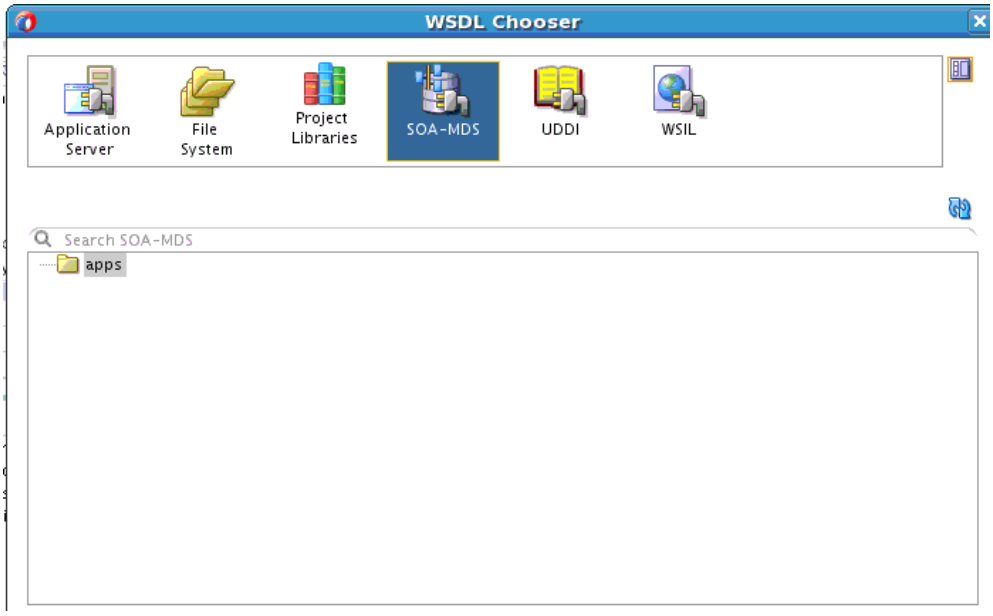
2. Select the Release 11*g* MDS Repository to export to a JAR file.

3. To import the JAR file into a Release 12*c* design-time MDS Repository, see section Importing the Contents of the JAR File into the /apps Folder.

Browsing for Files in the SOA Design-Time MDS Repository

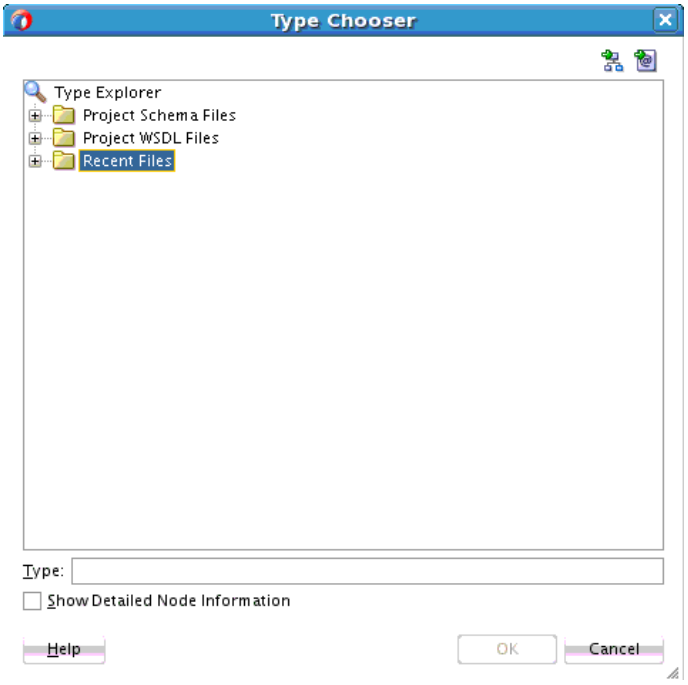
You can browse for and select files in the SOA Design-Time MDS Repository. For example, the WSDL Chooser dialog that you access from the Create Web Service dialog includes a selection for the SOA Design-Time MDS Repository, as shown in Figure 3-18.

Figure 3-18 SOA-MDS Selection in the WSDL Chooser Dialog



The Type Chooser dialog includes a **Recent Files** folder in which information is kept for the duration of the Oracle JDeveloper session. For example, if you create a new BPEL process and want to define the input variable from a schema in the SOA Design-Time MDS Repository, you go there once. When you want to define the output variable from the same schema, the schema remains visible in the **Recent Files** folder. Figure 3-19 shows the **Recent Files** folder.

Figure 3-19 Type Chooser



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. BPEL process creation and validation are described, along with key BPEL design features such as activities, partner links, adapters, and monitors.

This chapter includes the following sections:

Introduction to the BPEL Process Service Component

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

How to Add a BPEL Process Service Component

You add BPEL process service components to SOA composite applications in the .

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	a. Click *JDev_Oracle_Home*`\jdeveloper\JDev\bin\jdev.exe` or create a shortcut.	a. Go to `$ORACLE_HOME/jdeveloper/jdev/bin/`. b. Execute the following command: `./jdev`

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

   As a service component in an existing SOA composite application:

From the **Components** section of the Components window, drag a **BPEL Process** service component into the . This invokes the Create BPEL Process dialog shown in Figure 4-1.

In a new application:

a. From the Applications window, select **File** > **New** > **Application**.

b. Under **General** in the **Categories** list, select **Applications**.

c. In the **Items** list, select **SOA Application**, and click **OK**.

This starts the Create SOA Application wizard.

d. In the Application Name dialog, enter an application name in the **Application Name** field.

e. In the **Directory** field, accept the default location or enter a new directory path in which to create the SOA composite application.

f. Click **Next**.

g. In the Project Name dialog, enter a name in the **Project Name** field.

h. In the **Directory** field, accept the default location or enter a new directory path in which to create the project.

i. Click **Next**.

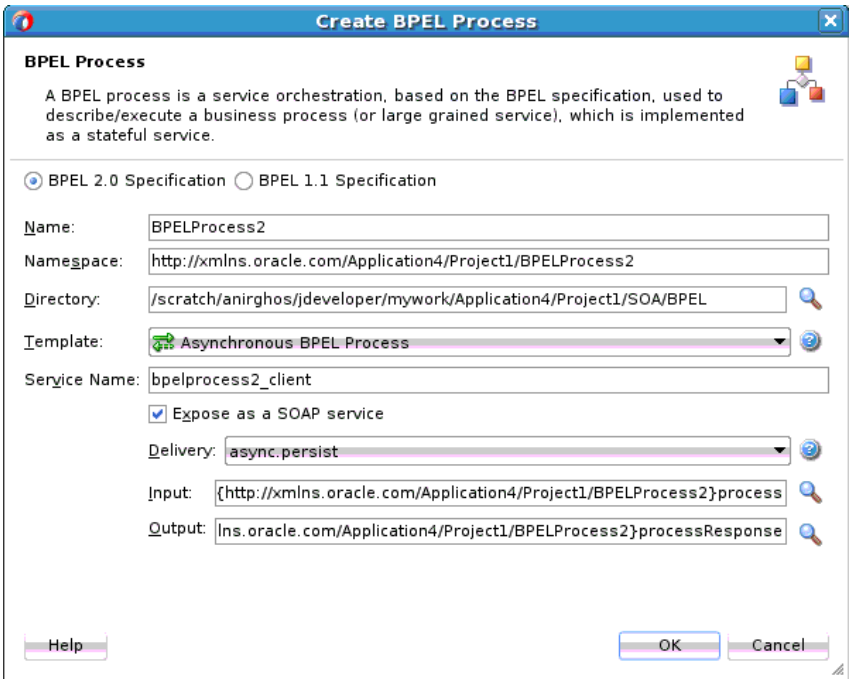
j. In the **Start from** section, ensure that **Standard Composite** is selected. The other selection, **SOA Template**, enables you to create a reusable part of a SOA project to bootstrap new projects. For more information, see Oracle SOA Suite Templates and Reusable Subprocesses .

k. In the Project SOA Settings dialog, select **Composite With BPEL Process**.

l. Click **Finish**.

This invokes the Create BPEL Process dialog shown in Figure 4-1.

Figure 4-1 Create BPEL Process Dialog



3. Provide the required details, as described in Table 4-2.

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.

Table 4-2 Create BPEL Process Dialog

Field	Description
BPEL Specification	Select the type of BPEL process to create. • **BPEL 2.0 Specification** Creates a BPEL project that supports the BPEL 2.0 specification. This is the default selection. • **BPEL 1.1 Specification** Creates a BPEL project that supports the BPEL 1.1 specification.

Table 4-2 (Cont.) Create BPEL Process Dialog

Field	Description
Name	Enter a name for the BPEL process or accept the default name. The name you enter becomes the file name for the BPEL process and Web Services Description Language (WSDL) files in the Applications window. Always use completely unique names when creating BPEL processes. Do not create the following: • 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.
Namespace	Use the default namespace path or enter a custom path.
Directory	Specify a directory in which to place BPEL process service component artifacts or accept the default directory of *`project_root_directory`*`/SOA/BPEL`. You can change the directory path, but ensure that the directory is beneath the `SOA` folder (that is, *`project_root_directory`*`/SOA`). If you specify a directory outside of `SOA`, an error message is displayed and the BPEL process is not created.

Table 4-2 (Cont.) Create BPEL Process Dialog

Field	Description
Template	Select a template based on the type of BPEL process service component you want to design. A template provides a basic set of default files in the Applications window (*process_name*.wsdl and *process_name*.bpel) with which to begin designing your BPEL process service component. • **Asynchronous BPEL Process**: Creates an asynchronous process with a default receive activity to initiate the BPEL process service component flow and an invoke activity to asynchronously call back the client. This type is selected by default. For more information, see Invoking an Asynchronous Web Service from a BPEL Process. • **Synchronous BPEL Process**: Creates a synchronous process with a default receive activity to initiate the BPEL process service component flow and a reply activity to return the results. For more information, see Invoking a Synchronous Web Service from a BPEL Process. • **One Way BPEL Process**: Creates a process with a one-way call interface definition. • **Define Service Later**: Select to create an empty BPEL process service component with no activities. • **Base on a WSDL**: Creates a BPEL process with an interface defined by an existing WSDL file. You must specify the WSDL Uniform Resource Locator (URL), port type, and callback port type to use. • **Subscribe to Events**: Creates a BPEL process in which you can subscribe to a business event. After selecting this option, the dialog refreshes to display an event table. Click the **Add** icon to select an event to which to subscribe. Your selection is then displayed in the event table. You can then select the consistency level and whether to publish this event. You can also click the **Filter** icon to create a filter expression for the selected event. This selection launches the Expression Builder dialog. For more information, see Using Business Events and the Event Delivery Network.
Service Name	Accept the default value or enter the name of the service this process is exposing. When you open an invoke, receive, OnMessage, or reply activity, the service name appears by default in the **Partner Link** field. This name is the same name as the partner link.
Expose as a SOAP Service	Select this check box to create a BPEL process service component that is automatically connected (wired) to an inbound simple object access protocol (SOAP) web service binding component. If you do not select this check box, the BPEL process service component is created as a standalone component in the SOA Composite Editor. You can explicitly associate the BPEL process service component with a service at a later time. This check box is selected by default.

Table 4-2 (Cont.) Create BPEL Process Dialog

Field	Description
Delivery **Note**: This field is displayed if you selected one of these templates in the **Template** list: • **Asynchronous BPEL Process** • **One Way BPEL Process** • **Subscribe to Events**	Set the persistence policy of the process in the delivery layer. This list enables you to specify a value for the `oneWayDeliveryPolicy` deployment descriptor property. The possible values are: • **async.persist**: Messages are persisted in the database. With this setting, reliability is obtained with some performance impact on the database. In some cases, overall system performance can be impacted. This is the default value. • **async.cache**: Incoming delivery messages are kept only in the in-memory cache. If performance is preferred over reliability, consider this setting. When set to **async.cache**, if the rate at which one-way messages arrive is much higher than the rate at which they are delivered, or if the server fails, messages can be lost. In addition, the system can become overloaded (messages become backlogged in the scheduled queue) and you can receive out-of-memory errors. Consult your own use case scenarios to determine if this setting is appropriate. When you set `oneWayDeliveryPolicy` to `async.cache` in high availability environments, invoke and callback messages in the middle of execution at the time of a server crash may be lost or duplicated. Server failover is not supported for `async.cache`. For more information, see *High Availability Guide*. • **sync**: Direct invocation occurs on the same thread. The scheduling of messages in the invoke queue is bypassed, and the BPEL instance is invoked synchronously. In some cases this setting can improve database performance. For information about transaction and fault propagation semantics for this property, see Transaction and Fault Propagation Semantics in BPEL Processes. For information about changing the value of this property in the Property Inspector, see How to Define Deployment Descriptor Properties in the Property Inspector.

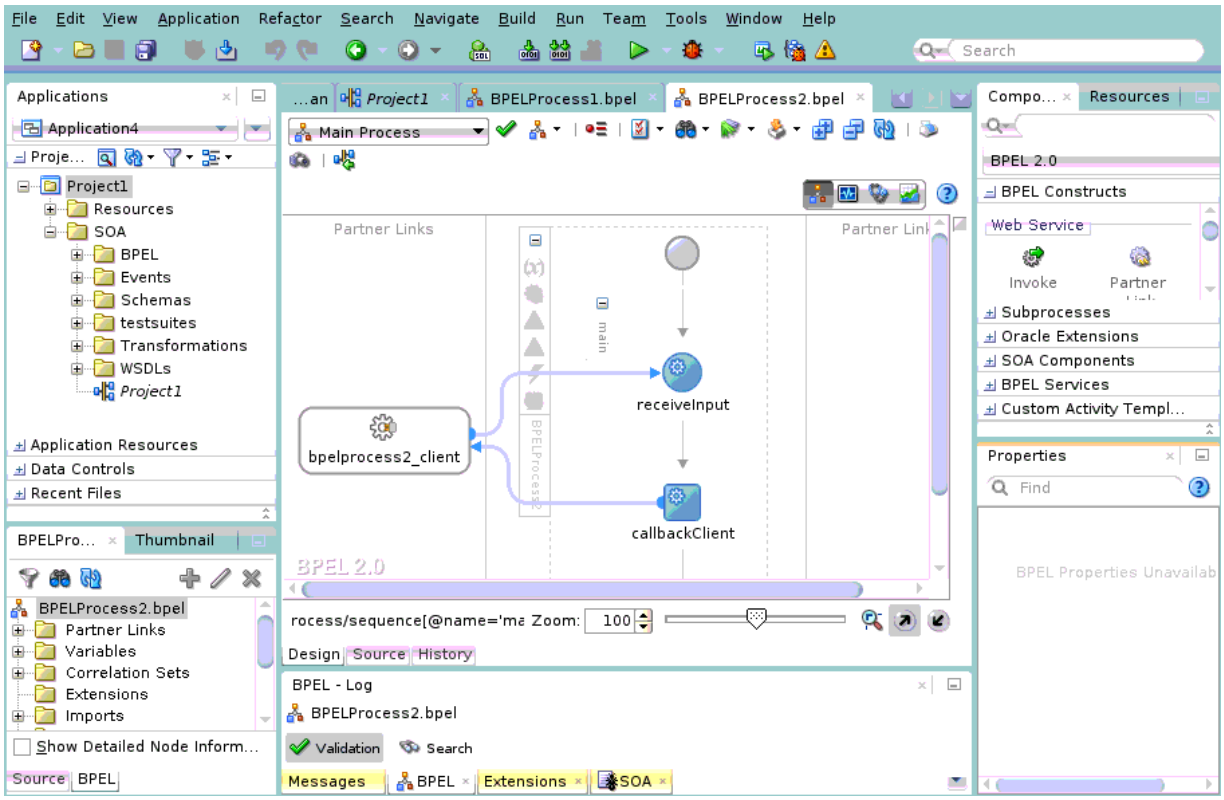
Table 4-2 (Cont.) Create BPEL Process Dialog

Field	Description
Transaction **Note**: This field is displayed if you selected **Synchronous BPEL Process** in the **Template** list.	Set the transaction behavior of the BPEL instance for initiating calls. This list enables you to specify a value for the `transaction` deployment descriptor property. The possible values are: • **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 the **Delivery** list value (`oneWayDeliveryPolicy` property) is set to **sync**, the invoke message is processed using the same thread in the same transaction. This is the default value. • **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 the **Delivery** list value (`oneWayDeliveryPolicy` property) is set to **sync**. • **notSupported**: Enables activities of business processes to be executed without a 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 a correlation is needed and it is always done asynchronously. For information about transaction and fault propagation semantics for this property, see Transaction and Fault Propagation Semantics in BPEL Processes. For information about changing the value of this property in the Property Inspector, see How to Define Deployment Descriptor Properties in the Property Inspector.
Input	Accept the default input XSD schema or click the **Search** icon to select a different XSD. If you click the **Search** icon, the Type Chooser dialog appears. Browse the imported schemas and select the input element (for example, a purchase order). You can also import an existing schema or WSDL in the Type Chooser dialog. The Type Chooser dialog displays information based on the context of its use. For example, if selecting a simple, message, or element type for a variable, the dialog displays XML schema simple types, WSDL file message types, or XML schema elements, respectively. If selecting a message part type, the dialog displays project schema files, XML schema simple types, and project WSDL files.
Output	Accept the default output XSD schema or click the **Search** icon to select a different XSD. If you click the **Search** icon, the Type Chooser dialog appears. Browse the imported schemas and select the output element (for example, a purchase order).

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 4-3 identifies the sections listed in Figure 4-2.

Table 4-3 Oracle JDeveloper Sections

Element	Description
Applications window (Upper left)	Displays the directories and files of a SOA project. Key directories and files beneath the **SOA** folder include the following: • **BPEL** Displays the BPEL process service component file (**.bpel**). • **Events** Displays the business event files (**.edn**). • **Schemas** Displays the BPEL process schema files. • **testsuites** Displays the test suite files. For more information, see Automating Testing of SOA Composite Applications. • **Transformations** Displays the transformation XSLT (**.xsl**) and XQuery (**.xqy**) mapper files. • **WSDLs** Displays the BPEL process WSDL files. • ***composite_name*** Describes the entire SOA composite application (sometimes referred to as the **composite.xml** file). For more information about this file, see What Happens When You Create a SOA Application and Project.
Oracle BPEL Designer (Design tab)	Provides a graphical 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 Applications window. • Click the **Design** tab at the bottom of the designer 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.

Table 4-3 (Cont.) Oracle JDeveloper Sections

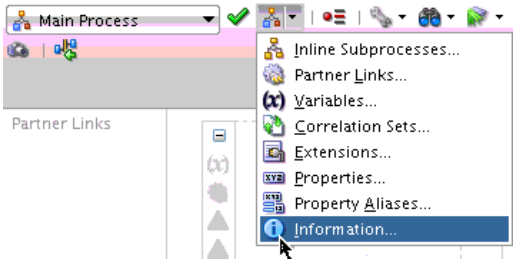
Element	Description
Components window (Upper right)	Displays the available activities to add to the BPEL process service component. Activities are the building blocks. The **BPEL Constructs**, **Subprocesses** (initially empty), and **Oracle Extensions** selections of the Components window display a set of activities and subprocesses that you drag into the designer of the BPEL process service component. The Components window displays only those pages relevant to the state of the designer. **BPEL Constructs**, **Subprocesses**, and **Oracle Extensions** are nearly always visible. However, if you are designing a transformation in a transform activity, the Components window only displays selections relevant to that activity, such as **String Functions**, **Mathematical Functions**, and **Node-set Functions**.
Structure window (Lower left)	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 (Lower middle)	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 tab	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 tab	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 open it for editing. For more information, see How to Edit BPEL Activities in the Property Inspector.

Note:

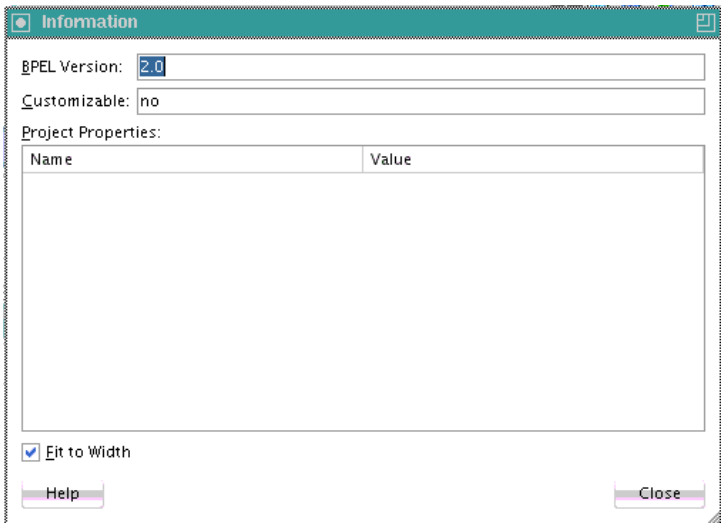
To learn more about these sections, you can also place the cursor in the appropriate section and press **F1** to display online Help.

5. Select **Information** from the **Property Structure** list 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



The Information dialog is displayed.



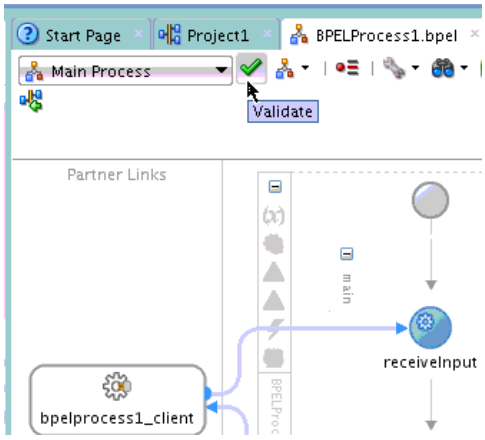
How to Validate a BPEL Process Service Component

You can syntactically and semantically (for example, the partner links or variables are not defined in an invoke activity) validate a BPEL process. If validation fails, information is displayed in the Log window.

To validate a BPEL process service component:

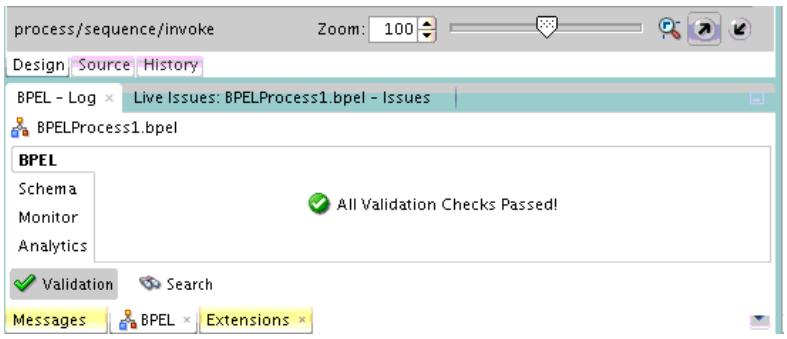
1. In Oracle BPEL Designer, click the green checkmark icon above the BPEL process. Figure 4-4 provides details.

Figure 4-4 Validation Icon in Oracle BPEL Designer



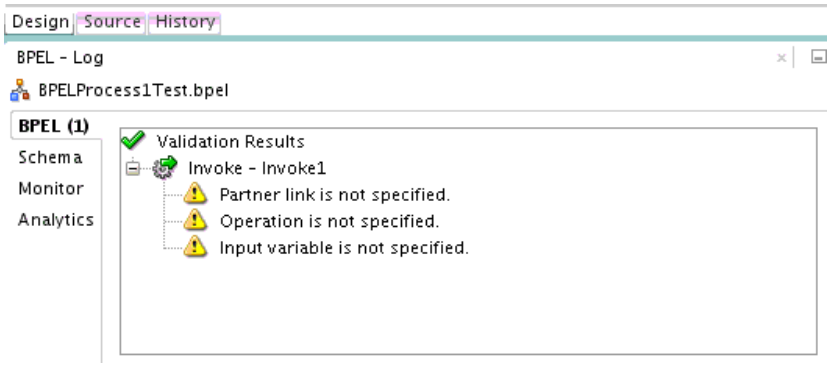
2. View the validation results in the Log window, as shown in Figure 4-5.

Figure 4-5 BPEL Process Validation Results in Log Window



If validation errors occur, messages are displayed in the Log window, as shown in Figure 4-6.

Figure 4-6 Log Window Validation Results



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 4-7 shows an assign activity.

Figure 4-7 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 4-8 shows an invoke activity.

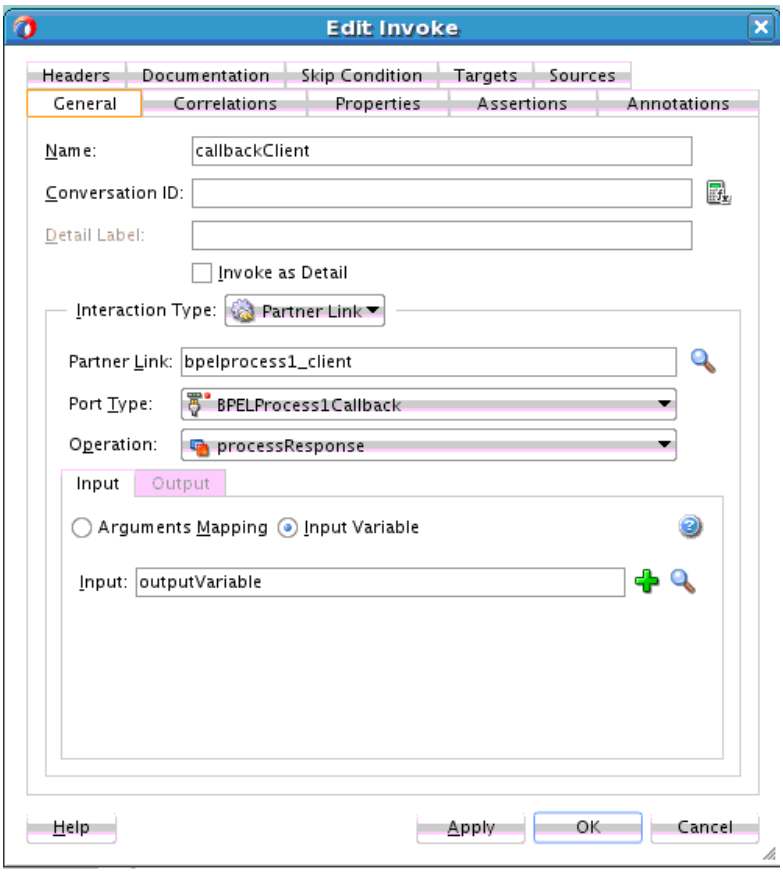
Figure 4-8 Invoke Activity

- A receive activity waits for an asynchronous callback response message from a service. Figure 4-9 shows a receive activity. A receive activity is also used when a process is started asynchronously through a partner link.

Figure 4-9 Receive Activity

Figure 4-10 shows an example of a property window (for this example, an invoke activity).

Figure 4-10 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 BPEL Process Activities and Services.

For information about copying and pasting activities in the same project or between projects, see How to Copy and Paste Activities in BPEL Projects .

For information about editing activities in the Property Inspector, see How to Edit BPEL Activities in the Property Inspector.

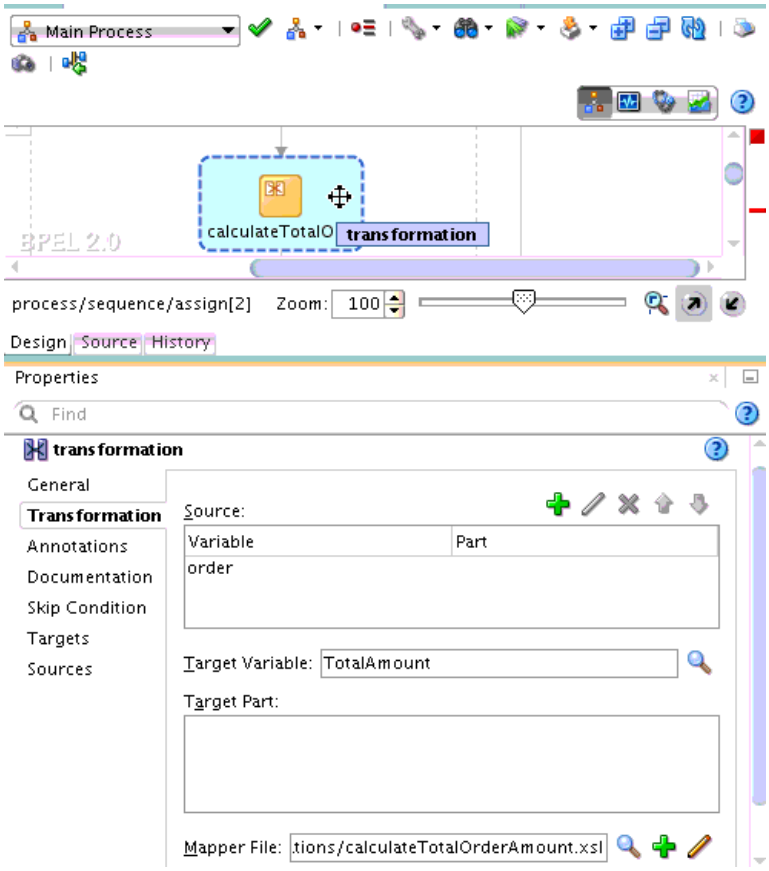
How to Edit BPEL Activities in the Property Inspector

You can edit the property fields of activities in BPEL 1.1 and 2.0 processes in the Property Inspector of Oracle BPEL Designer in Oracle JDeveloper. This action is the same as double-clicking an activity or right-clicking an activity and selecting **Edit**, making changes, and clicking **Apply** or **OK**.

To edit BPEL activities in the Property Inspector:

1. In Oracle BPEL Designer, single-click an activity. For this example, an XSLT transform activity is selected in Figure 4-11.
2. The property fields of the activity are displayed for editing in the Property Inspector below Oracle BPEL Designer.

Figure 4-11 Activity is Displayed for Editing in the Property Inspector

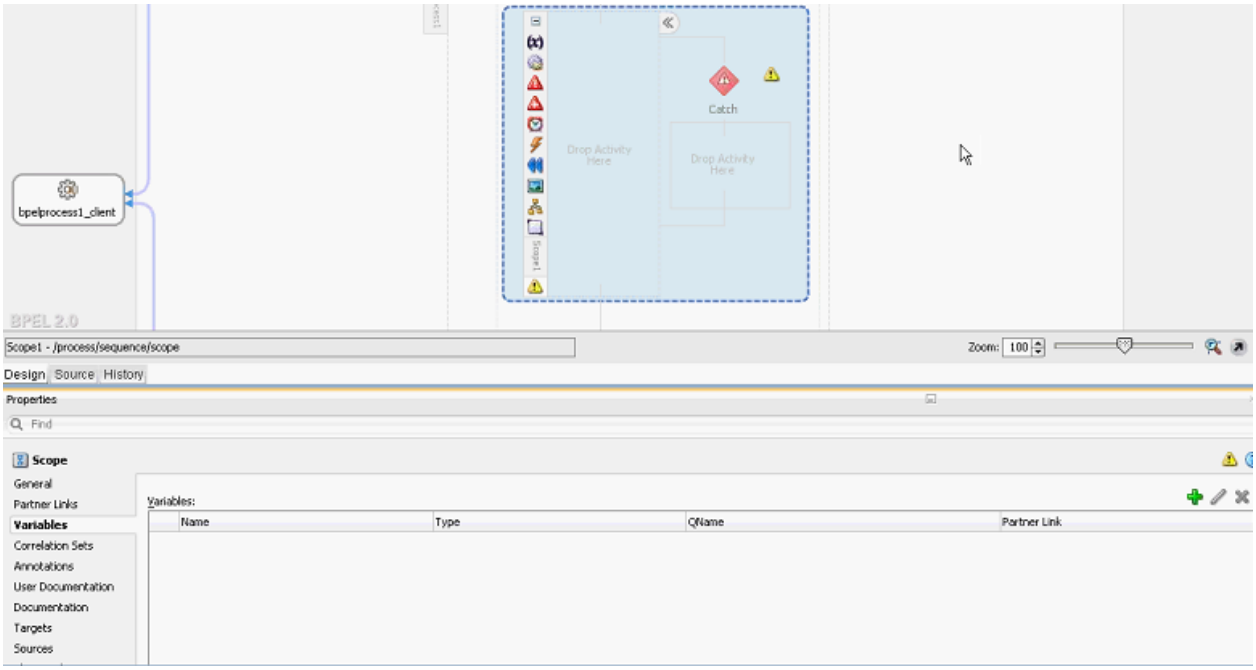


3. Make changes and press the **Apply** key, or navigate away from the activity by clicking another activity.
4. Return to the activity you edited and note that the changes have been applied.

   You can also edit the actions within a scope activity, such as catch activities, variable, and so on.
5. Expand a scope activity.
6. In the Property Inspector, click **Variables**.

   The Property Inspector is refreshed to display the property fields for a variable, including the **Add**, **Edit**, and **Delete** icons. Figure 4-12 provides details.

Figure 4-12 Variable Section of a Scope Activity is Displayed for Editing in the Property Inspector



How to Copy and Paste Activities in BPEL Projects

You can copy and paste activities in the same BPEL project or between BPEL projects. This prevents you from having to create similar activities from start to finish multiple times. You can design an activity once and use it in multiple places, editing it as necessary.

Note:

You can copy an individual OnAlarm activity from one scope activity and paste it into another scope activity. You can also copy an individual OnAlarm activity from one pick activity and paste it into another pick activity.

Note the following restrictions:

- You cannot copy activities from a BPEL 1.1 project to a BPEL 2.0 project or from a BPEL 2.0 project to a BPEL 1.1 project.
- In BPEL 2.0 projects, you cannot copy an individual OnAlarm activity from a pick activity into a scope activity, or vice versa. However, this type of copying and pasting is supported in BPEL 1.1 projects.
- When you copy and paste a scope activity, the variables referenced in the first scope activity are not copied.

To copy and paste activities:

1. Right-click the activity to copy.
2. Select **Copy**.
3. Go to the project in which to paste the activity.

4. Perform one of the following tasks:

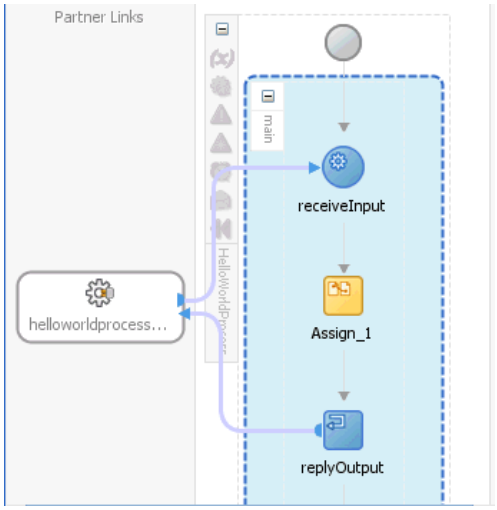
   a. Right-click the activity closest to where you want to paste the activity.

   b. Choose to either paste the activity before or after the selected activity.

   or

   a. Highlight the BPEL process, as shown in Figure 4-13.

   Figure 4-13 Selected BPEL Process



   b. Right-click and select **Paste** > **Paste Into**.

   The activity is pasted at the top of the BPEL process.

How to Add a Description of Actions to BPEL Process Activities

You can add a description of actions to a BPEL process activity. This creates a **TODO Tasks** icon on the activity. When you place your cursor over this icon, it displays the description of actions. The description can describe the actions performed by the activity in the BPEL process service component.

How to add a description of actions to BPEL process activities:

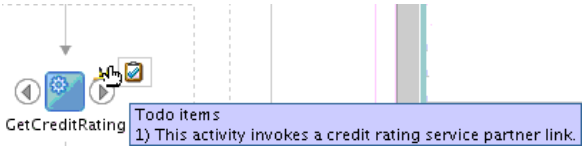
1. Right-click an activity, and select **Add TODO Task**.

   The Add TODO Task dialog is displayed.

2. Add a description of the actions performed by the activity, then click **OK**.

3. Place the cursor over the **TODO Tasks** icon to the right of the BPEL activity to display the description. Figure 4-14 provides details.

Figure 4-14 Description of BPEL Activity



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 (the recommended method) or within a BPEL process service component in Oracle BPEL Designer. Figure 4-15 shows the partner link icon (for this example, named **PartnerSupplierMediator**).

Figure 4-15 Partner Link 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 4-16 shows an example of the attributes of a partner link for a service.

Figure 4-16 Partner Link Dialog

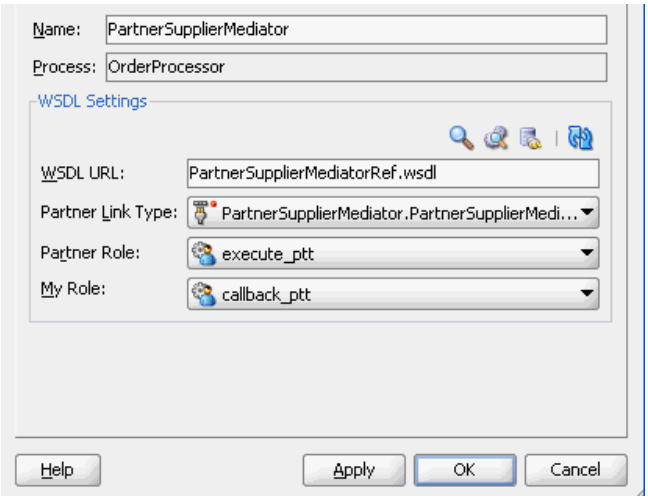


Table 4-4 describes the fields of this dialog.

Table 4-4 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.

Table 4-4 (Cont.) Create Partner Link Dialog Fields

Field	Description
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 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. If this is a synchronous process case, the BPEL process service component does not have a role.

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.

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.

For information about editing partner links in the Property Inspector, see How to Edit BPEL Activities in the Property Inspector.

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 in the . 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.

How to Create a Partner Link

To create a partner link:

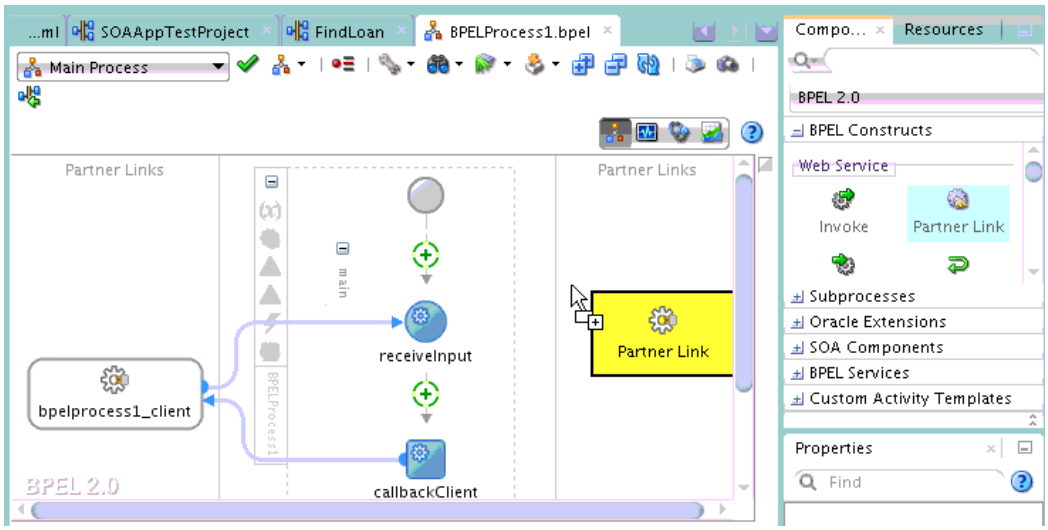
1. In the SOA Composite Editor, double-click the BPEL process service component.

   Oracle BPEL Designer is displayed.

2. In the Components window, expand **BPEL Constructs**.

3. Drag a **Partner Link** into the appropriate **Partner Links** swimlane, as shown in Figure 4-17.

   Figure 4-17 Partner Link Creation in Oracle BPEL Designer



   The Create Partner Link dialog appears.

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

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

Partner Links for an Outbound Adapter

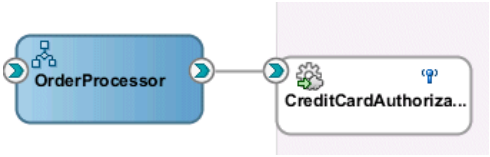
Table 4-5 describes the impact on the .

Table 4-5 Impact of Partner Link Creation on the

Creating the Following for a BPEL Process in Oracle BPEL Designer...	Displays the Following in the ...
A partner link for an *outbound* adapter	• A reference handle for the BPEL process service component • A reference representing the outbound adapter in the composite • A wire connecting the BPEL process service component to the adapter reference

Figure 4-18 shows how this method of creation appears in the .

Figure 4-18 Impact



Partner Links for an Inbound Adapter

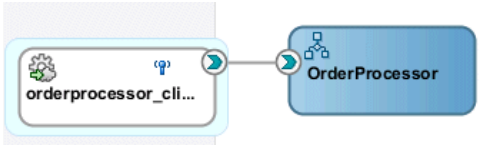
Table 4-6 describes the impact on the .

Table 4-6 Impact of Partner Link Creation on the

Creating the Following for a BPEL Process in Oracle BPEL Designer...	Displays the Following in the ...
A partner link for an *inbound* adapter	• A service for the BPEL process service component • A service representing the inbound adapter in the composite • A wire connecting the inbound adapter service to the BPEL process service component

Figure 4-19 shows how this method of creation appears in the .

Figure 4-19 Impact



Partner Links from an Abstract WSDL to Call a Service

Table 4-7 describes the impact on the .

Table 4-7 Impact of Partner Link Creation on the

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

Partner Links from an Abstract WSDL to Implement a Service

Table 4-8 describes the impact on the .

Table 4-8 Impact of Partner Link Creation on the

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

Figure 4-20 shows how this method of creation appears in the .

Figure 4-20 Impact

Partner Links and Human Tasks or Business Rules

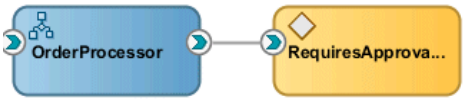
Table 4-9 describes the impact on the .

Table 4-9 Impact of Partner Link Creation on the

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

Figure 4-21 shows how this method of creation appears in the .

Figure 4-21 Impact



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

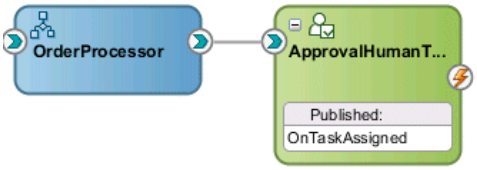
Table 4-10 describes the impact on the .

Table 4-10 Impact of Partner Link Creation on the

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

Figure 4-22 shows how this method of creation appears in the .

Figure 4-22 Impact



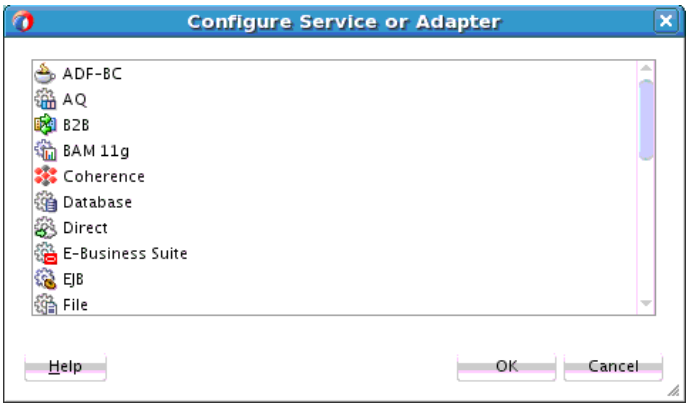
Introduction to Adapters

The Partner Link dialog shown in Figure 4-16 also enables you to take advantage of another key feature that and Oracle JDeveloper provide. Click the **Service Wizard** icon shown in Figure 4-23 to access the Adapter Configuration wizard.

Figure 4-23 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), Oracle User Messaging Service, and more. 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 4-24 for use with the BPEL process service component:

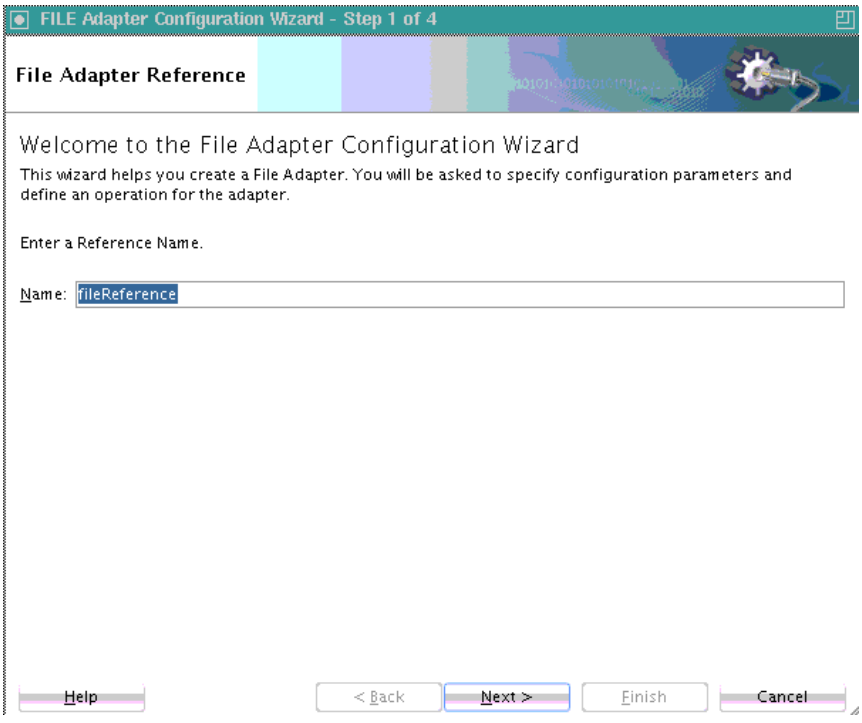
Figure 4-24 Service and Adapter Types



For information about the service and adapter types, see Getting Started with Binding Components.

When you select an adapter type (for this example, **File** was selected in Figure 4-24), the dialog shown in Figure 4-25 prompts you to enter a name. When the wizard completes, a WSDL file by this name appears in the Applications window under the **WSDLs** folder. 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 Applications window.

Figure 4-25 Adapter Service Name



The Adapter Configuration wizard dialogs that appear after the this dialog are based on the adapter type you selected.

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

For more information about technology adapters, see *Understanding Technology Adapters*.

Introduction to BPEL Process Monitors

You can configure BPEL process monitors in Oracle BPEL Designer by selecting **Change to Monitor view** at the top of Oracle BPEL Designer. Figure 4-26 provides details. BPEL process monitors can send data to Oracle BAM for analysis and graphical display through the Oracle BAM adapter.

Figure 4-26 BPEL Process Monitors

For information about business indicators, intervals, and counters, see the Oracle SOA Suite 11*g* documentation:

http://docs.oracle.com/cd/E28280_01/dev.1111/e10224/bam_adapter.htm#BABIJBCC

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, including one-way messages, synchronous and asynchronous interactions, one request - multiple and single responses, one request - mandatory and optional responses, partial processing, and multiple application interactions. It also describes the best use practices for each.

This chapter includes the following sections:

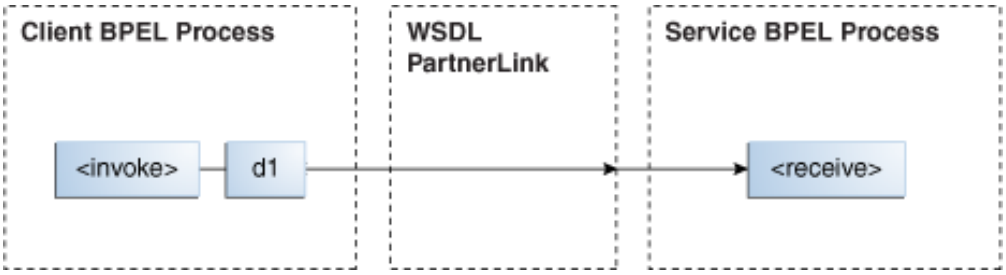
Introduction to One-Way Messages

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. The following example shows the `portType` and `operation` part of the BPEL process WSDL file for this environment.

```
. . .
<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.

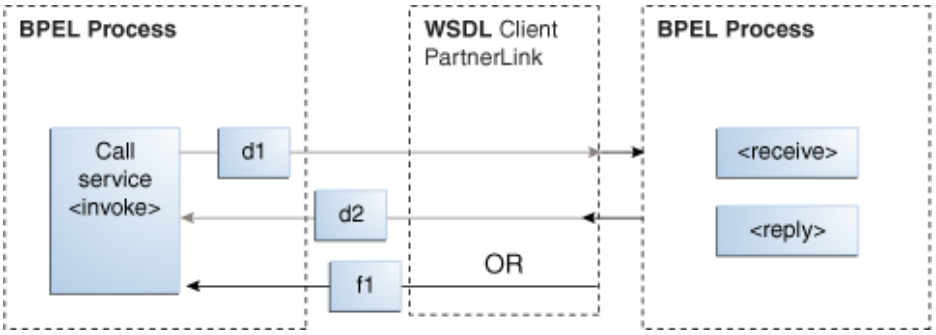
Introduction to Synchronous Interactions

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. The following example shows the `portType` and `operation` part of the BPEL process WSDL file for this environment.

```
. . .
<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 Invoking a Synchronous Web Service from a BPEL Process.

Synchronous BPEL Process Invoking an Asynchronous Process

If a synchronous BPEL process invokes an asynchronous process, the callback response message is not acknowledged by the BPEL process and the process times out waiting for a response. This type of interaction pattern is not supported.

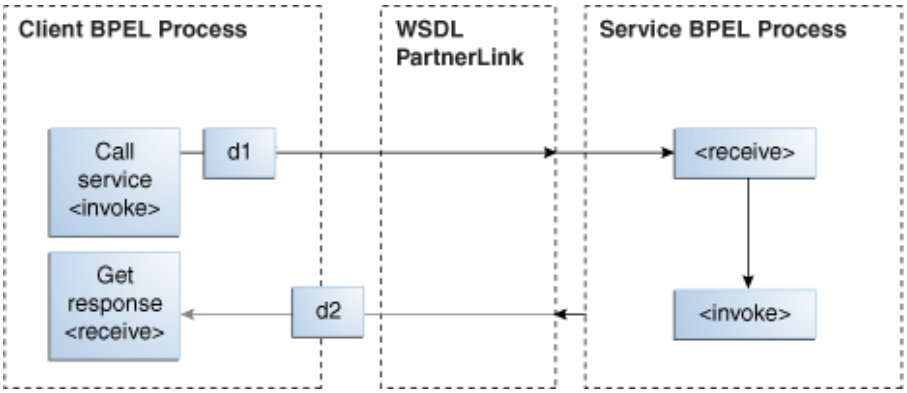
Introduction to Asynchronous Interactions

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

```
. . .
<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

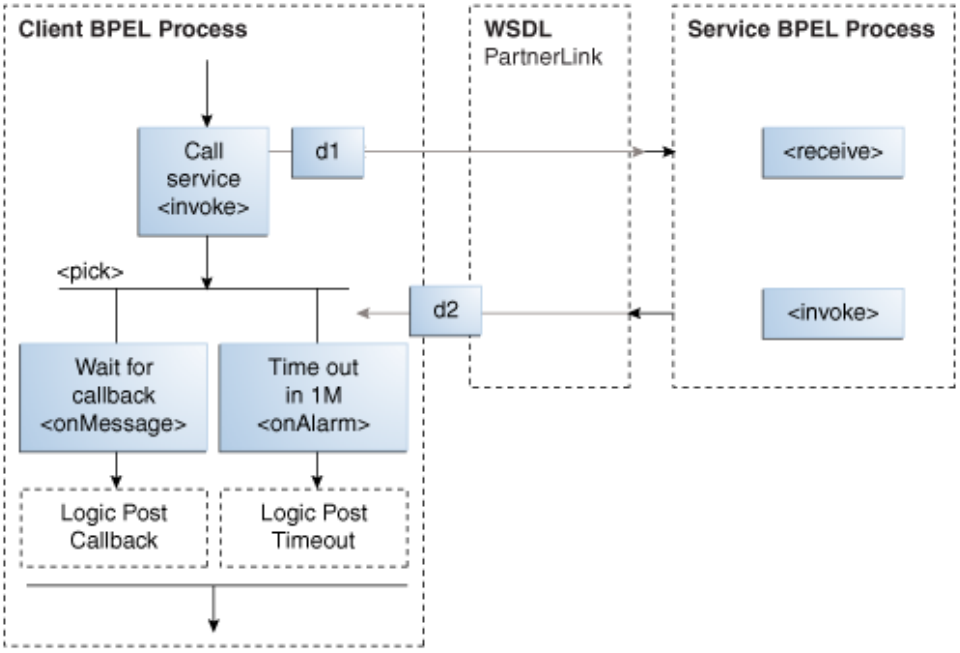
As with a synchronous transaction, when the BPEL process service component is on the service side of an asynchronous transaction, it needs a receive activity to accept the incoming request and an invoke activity to return either the requested information or a fault. Note the difference between this and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

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 Selecting Between Continuing or Waiting on a Process with a Pick Activity.

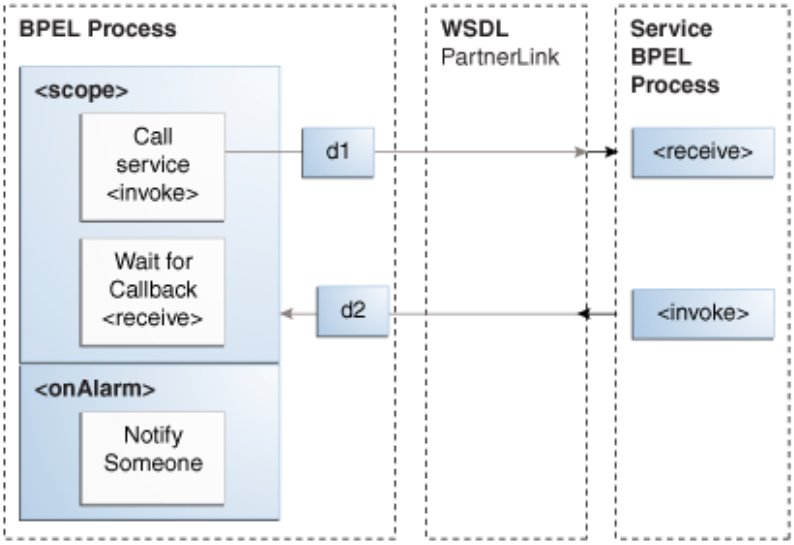
BPEL Process Service Component as the Service

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.

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.

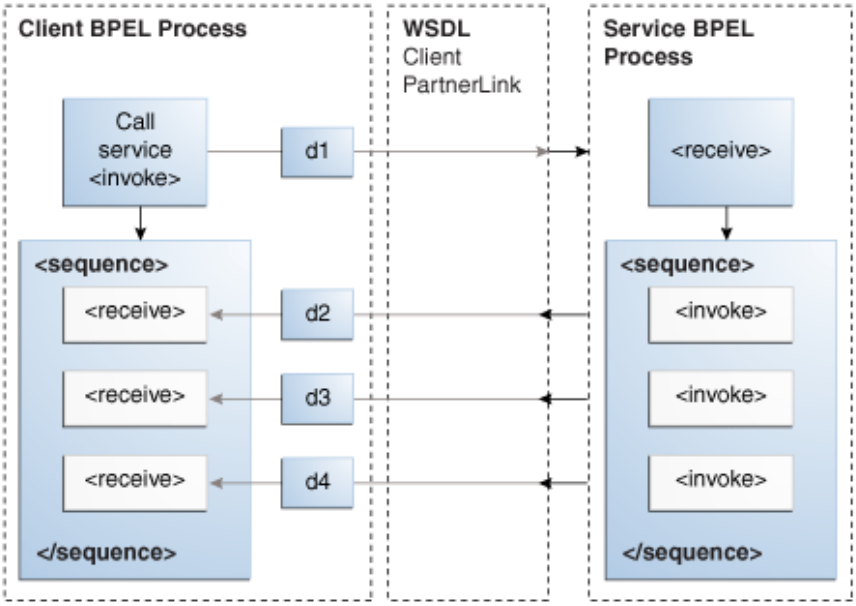
BPEL Process Service Component as the Service

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.

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.

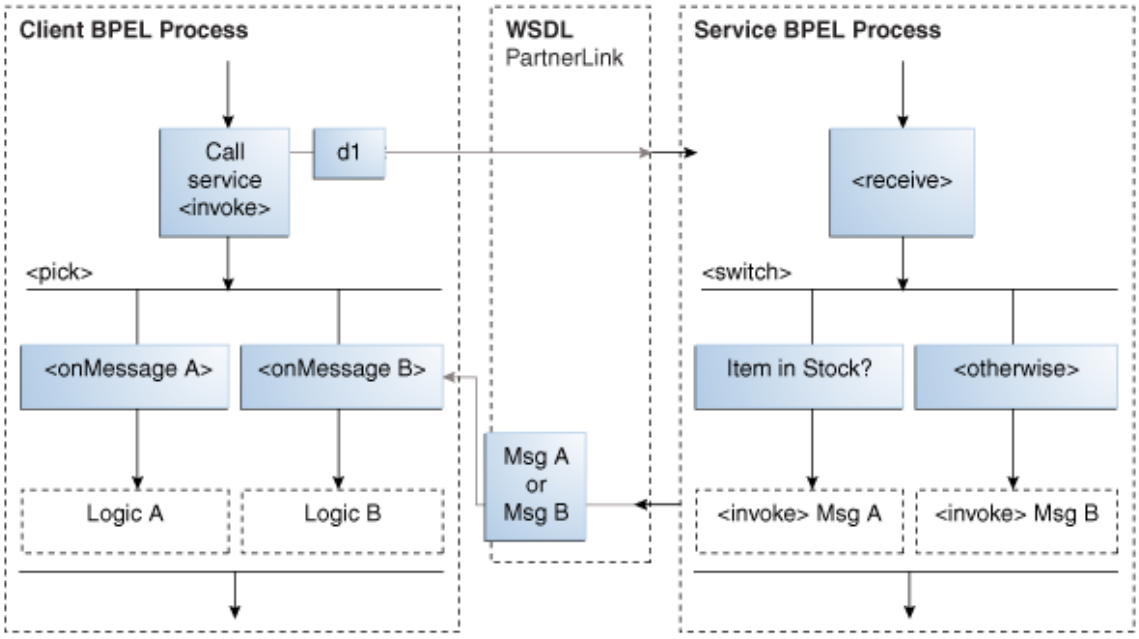
BPEL Process Service Component as the Service

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.

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 Selecting Between Continuing or Waiting on a Process with a Pick Activity.

BPEL Process Service Component as the Service

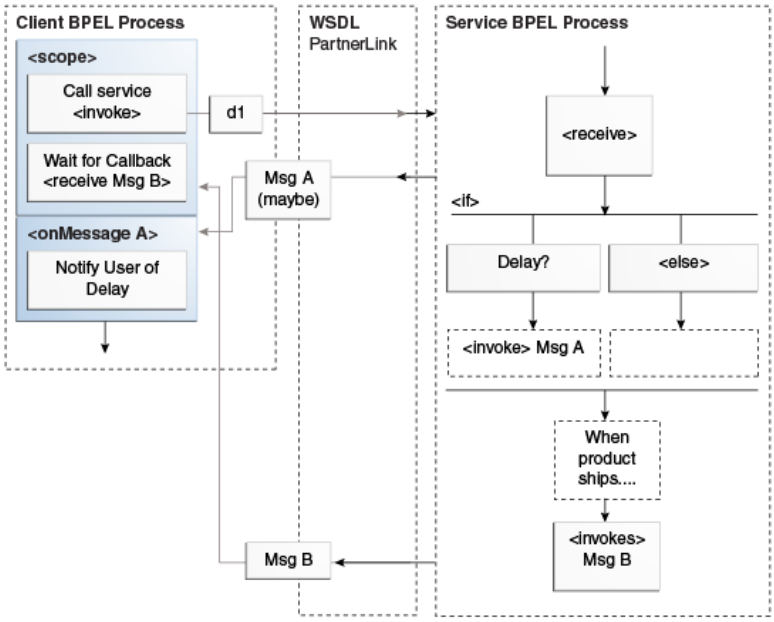
The BPEL service needs a receive activity to accept the message from the client, and a switch activity (in BPEL 1.1) or an if activity (in BPEL 2.0) with two branches, one with an invoke activity sending the in-stock message if the item is available, and a second branch with an invoke activity sending the out-of-stock message if the item is not available.

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.

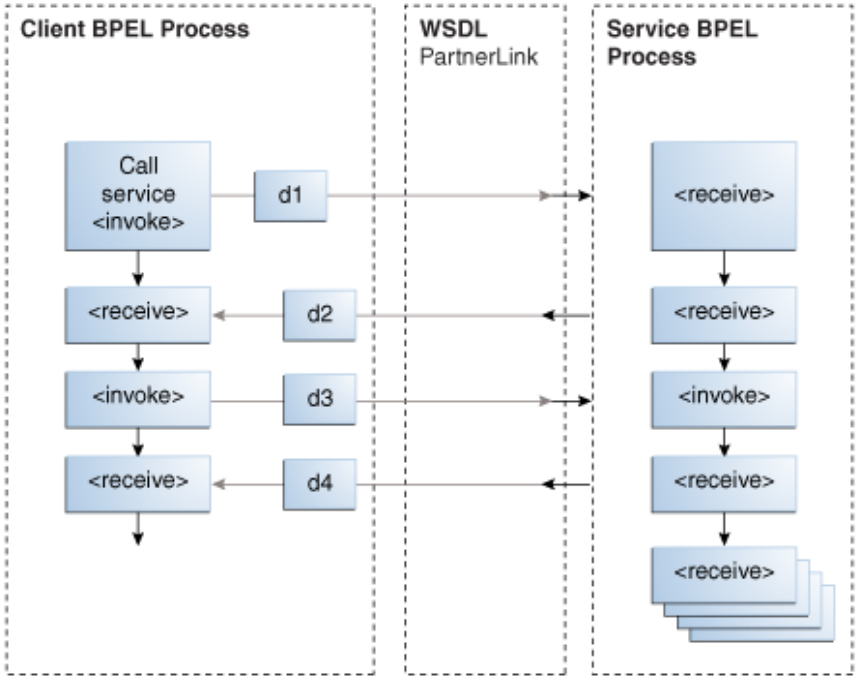
BPEL Process Service Component as the Service

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).

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

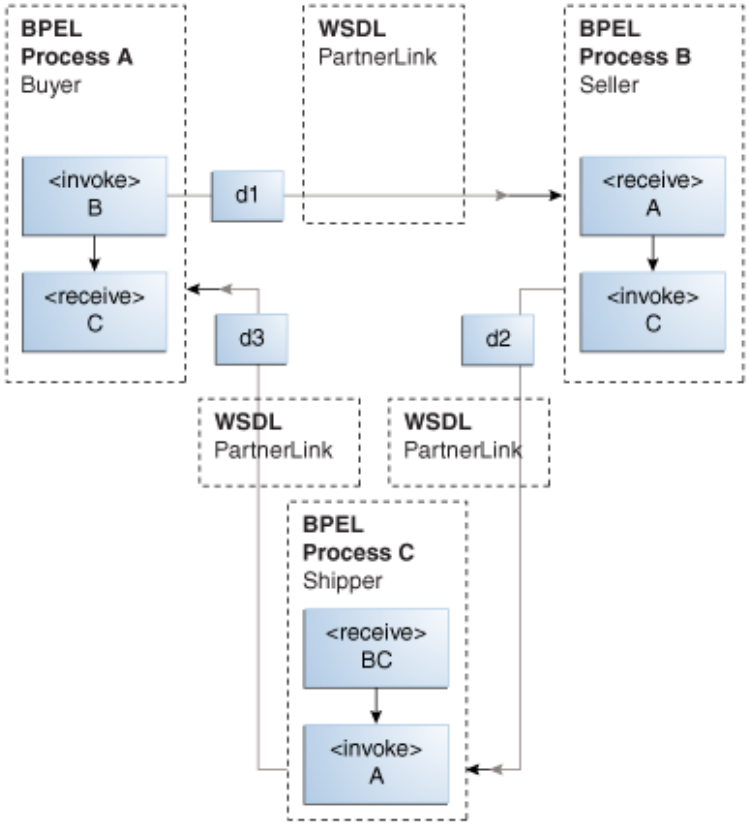
The BPEL service needs a receive activity for each request from the client, and an invoke activity for each response. Once the responses are finished, the BPEL process service component as the service can continue with its processing, using the information gathered in the interaction to perform the necessary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

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 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:

- Understanding Document-Style and RPC-Style WSDL Differences
- Manipulating SOAP Headers in BPEL
- Declaring Extension Namespaces in BPEL 2.0

Note:

Most of the examples in this chapter assume that the WSDL file defining the associated message types is document-literal style rather than the remote procedure call (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 Understanding Document-Style and RPC-Style WSDL Differences.

Introduction to Manipulating XML Data in BPEL Processes

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

XML Data in BPEL Processes

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.

Data Manipulation and XPath Standards in Assign Activities

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.

The following example shows the formal syntax for BPEL version 1.1, as described in the :

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

The next example shows the formal syntax for BPEL version 2.0, as described in the . 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.

```
<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 the following example:

```
<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. The following code provides an example:

  ```
  <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 (section 14.3) or (section 8.4), and the , 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 .

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 (such as `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 .

- 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 . For more information about `getVariableData`, see 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 . For more information about `getVariableProperty`, see 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` file and placed inside the `orabpel.jar` file. For more information, see Creating User-Defined XPath Extension Functions.

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

For more information about XPath and XQuery transformation code examples, see Creating Transformations with the XSLT Map Editor and Creating Transformations with the XQuery Mapper.

For more information about the assign activity, see Assign Activity.

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.

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.

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

Before Release 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 pre-11*g* 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 starting with Release 11*g* and continuing with Release 12*c*, 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 releases 11*g* and 12**c** with the entity variable.

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

10.1.*x* Releases	**11*g* and 12*c* Releases 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, a 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 Releases 11*g* and 12*c*.

How to Create an Entity Variable

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

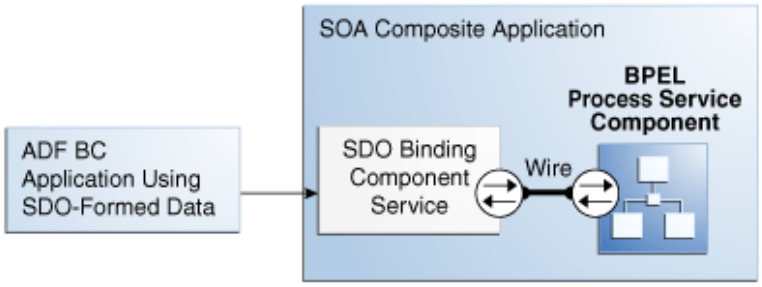
In Release 10.1.*x* of Oracle BPEL Process Manager, all variable data was in DOM format. Starting with Release 11*g* and continuing with Release 12*c*, variable data in SDO format is also supported. DOM and SDO variables in BPEL process service components are implicitly converted to the required forms. 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.

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 and Oracle BPEL Designer to perform the following tasks:

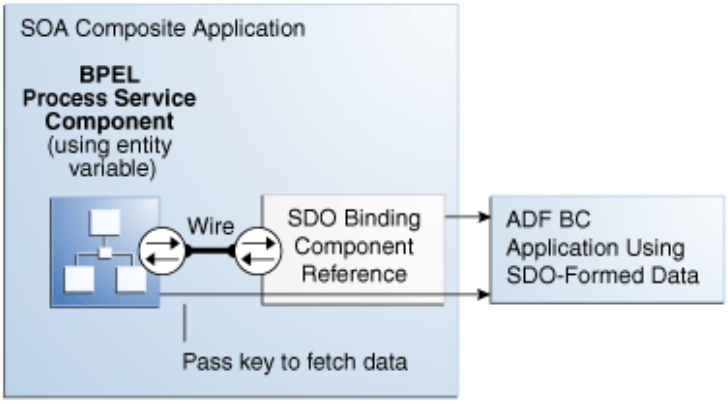
- Define an SDO binding component service and a BPEL process service component in the composite application.
- Connect (wire) the SDO service and BPEL process service component.
- Define the details of the BPEL process service component.

For more information about using the , see Getting Started with Developing SOA Composite Applications.

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 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.

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.

Note:

Entity variables are supported on BPEL projects that use version 1.1 or 2.0 of the BPEL specification.

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**.

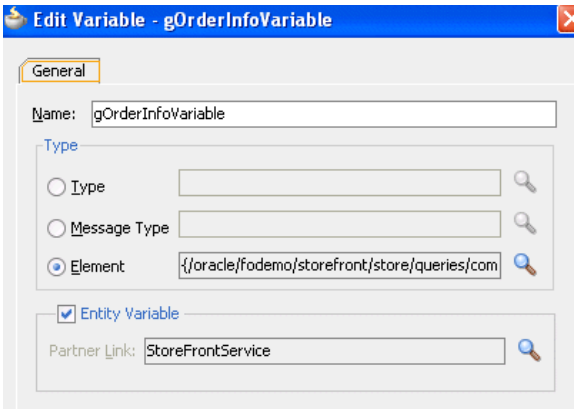
   The Create Variable dialog appears.

4. In the **Name** field, enter a name.
5. Click the **Entity Variable** check box 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**.
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



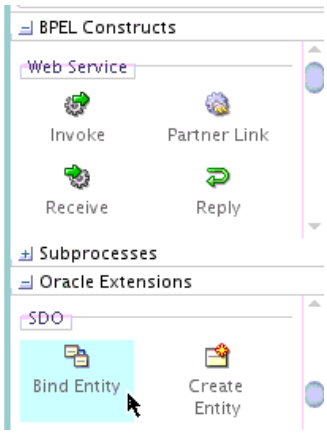
Creating a Binding Key

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 Components window, expand **Oracle Extensions**.
2. Scroll down to the **SDO** section.
3. Drag a **Bind Entity** activity into your BPEL process service component. Figure 6-4 provides details.

Figure 6-4 Bind Entity Activity in the Components Window



The Bind Entity dialog appears.

4. In the **Name** field, enter a name.
5. To the right of the **Entity Variable** field, click the **Search** icon.

   The Variable Chooser dialog appears.
6. Select the entity variable created in Creating an Entity Variable and Choosing a Partner Link and click **OK**.
7. In the **Unique Keys** section, click the **Add** icon.

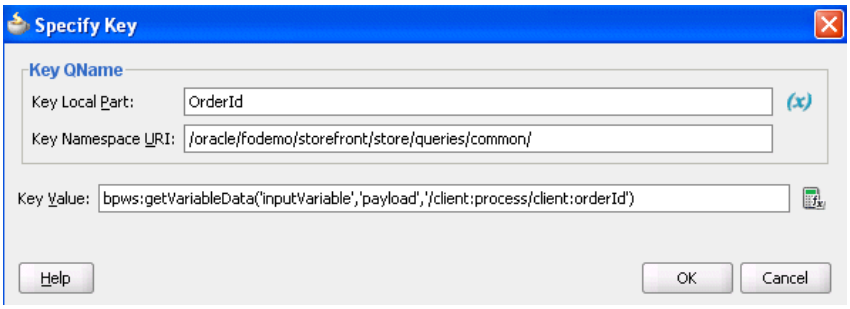
   The Specify Key dialog appears. You use this dialog to create a key for retrieving the order ID from the Oracle ADF Business Component data provider service.
8. Enter the details described in Table 6-2 to define the binding key:

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 6-5 shows the Specify Key dialog after completion.

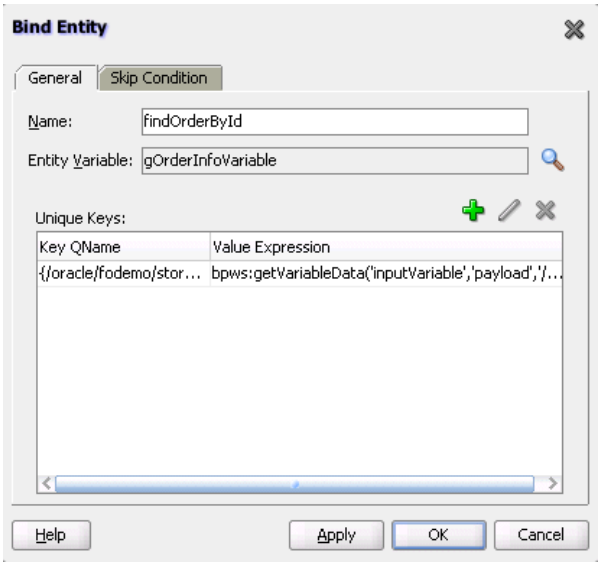
Figure 6-5 Specify Key Dialog



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

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

Figure 6-6 Bind Entity Dialog



10. 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 *Developing Fusion Web Applications with 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.

Translating Between Native Data and XML

The BPEL process translate activity enables you to translate messages between native XSD format and XML format. The following types of translation are supported:

- Inbound translation:
  - Native format to XML

  - Opaque to XML
  - Native to an attachment in a directory
- Outbound translation:
  - XML to native format
  - XML to an attachment in a directory
- Supported in both BPEL 1.1. and 2.0 projects.

Inbound message translation automatically uses the `doTranslateFromNative` function. Outbound message translation automatically uses the `doTranslateToNative` function). You do not need to create an assign activity and invoke the Expression Builder dialog to configure these functions. The translate activity automatically generates the assign activity.

How to Translate Native Data to XML Data

This section describes how to configure the translate activity in a BPEL process to receive an inbound message in native XSD format (for this example, string data) and translate it to XML format. The Native Format Builder wizard is used to create a new schema file.

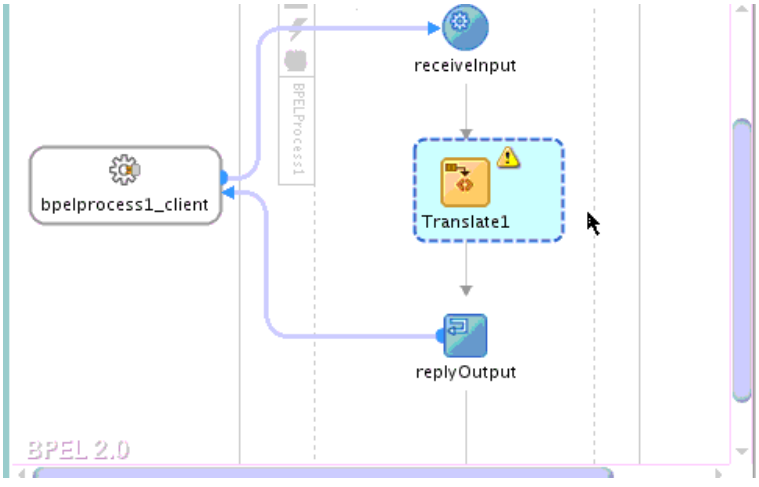
To translate native data to XML data:

1. Right-click a BPEL process in the SOA Composite Editor, and select **Edit**.

   Oracle BPEL Designer is displayed.

2. Expand the **Oracle Extensions** section of the Components window and drag a **Translate** activity into the BPEL process. Figure 6-7 provides details.

   Figure 6-7 Translate Activity in a BPEL Process

3. Right-click the translate activity and select **Edit**.

   The Translate dialog is displayed for editing.

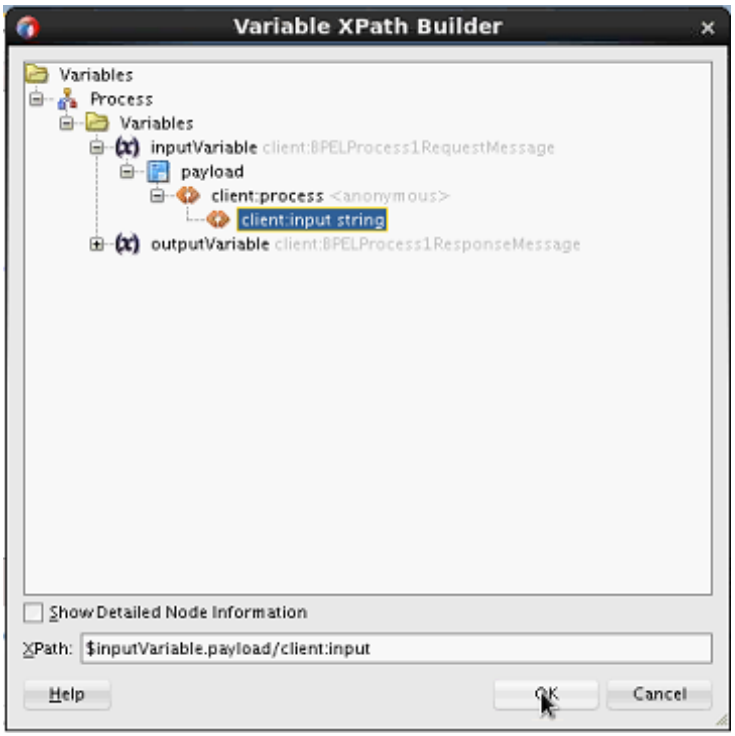
4. Select **Native to XML** to receive inbound native data (for this example, in a single string).

5. To the right of the **Input** field, click the **Browse** icon.

   The Variable XPath Builder dialog is displayed.

6. Select the native string that is part of the inbound payload to translate into XML format, and click **OK**. Figure 6-8 provides details.

Figure 6-8 Variable XPath Builder



7. To the right of the **NXSD Schema** field, select the schema to use:

   - If the schema already exists, select the **Search** (first) icon to invoke the Type Chooser dialog.
   - If the schema does not exist, select the second icon to invoke the Native Format Builder wizard to create the schema.

   The following example describes how to use the Native Format Builder wizard to create a new schema from a text file that uses a comma-separated delimiter.

   a. In the **File Name** field of the File Name and Directory dialog, enter a name, and click **Next**.

   b. In the Choose Type dialog, select **Delimited (Contains records whose fields are delimited by a special character)**, and click **Next**.

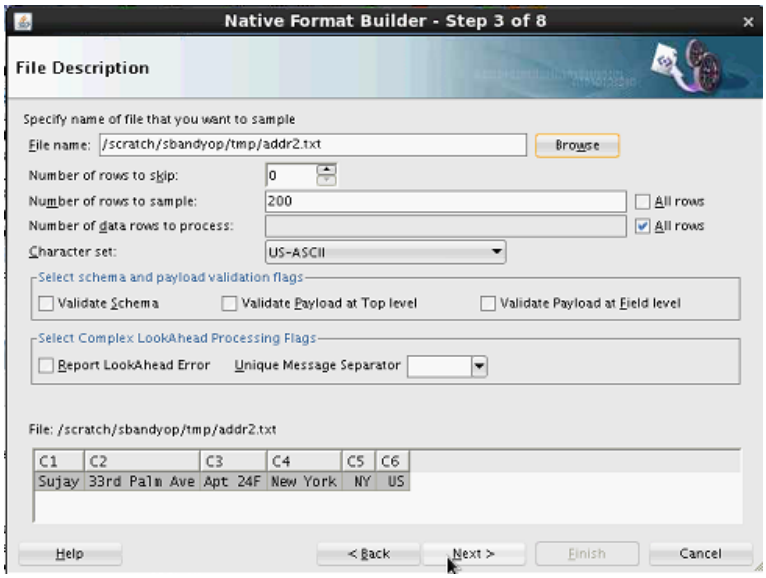
   c. In the File Description dialog, click **Browse** to select the text file that uses the comma-separated delimiter.

      The Select sample file dialog is displayed.

   d. Select the file to use, and click **OK**.

      The file contents are displayed at the bottom of the File Description dialog. Figure 6-9 provides details.

Figure 6-9 Sample File Contents



e. Click **Next**.

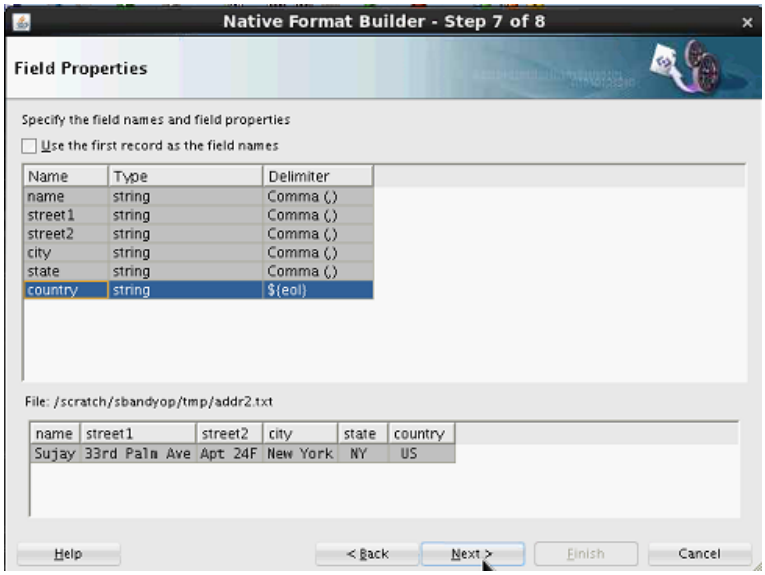
f. In the Record Organization dialog, click **Next**.

g. In the Specify Elements dialog, enter a name for the element to represent the record (for this example, `addr` is entered), and click **Next**.

h. In the Specify Delimiters dialog, accept the default value of a comma as the special character that delimits the fields in the text file, and click **Next**.

i. In the **Name** column of the Field Properties dialog, enter the appropriate values in place of **C1**, **C2**, **C3**, **C4**, **C5**, and **C6**, and click **Next**. Figure 6-10 provides details.

Figure 6-10 Name Column Default Values Replaced with Specific Values



The new schema is displayed in the Generated Native Format Schema dialog.

j. Click **Test** to test the schema.

k. In the **Result XML** section, click the green arrow.

The native schema and resulting XML are displayed. Figure 6-11 provides details.

Figure 6-11 Output From Testing the Native Schema

l. Click **OK** to return to the Generated Native Format Schema dialog.

m. Click **Next**, then **Finish**.

The **addr_schema1.xsd** file is created and displayed in the **NXSD Schema** field of the Translate dialog.

8. From the **Output Type** list, select **DOM**. Both DOM and SDOM supported if you select DOM.

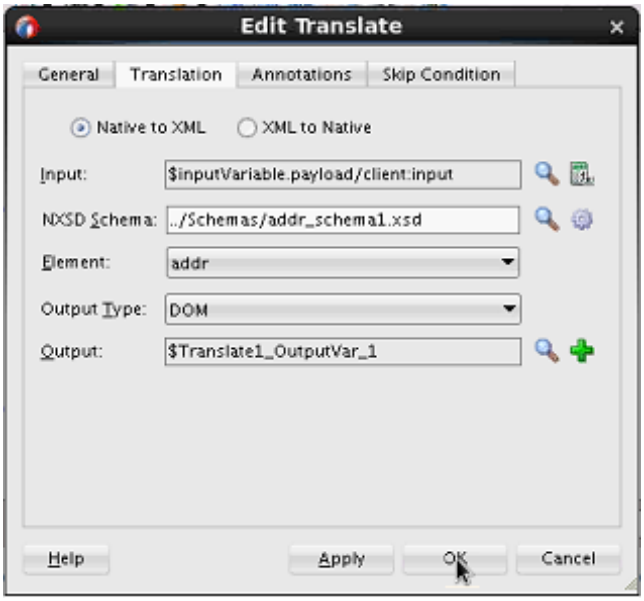
9. To the right of the **Output** field, select the variable for the schema.

a. If you have an output variable that adheres to the schema specified in Step 7, click the **Search** (first) icon to select the existing variable.

b. If you do not have an existing variable, click the **Add** (second) icon to invoke the Create Variable dialog. Accept the default values or rename the variable to create an output variable, and click **OK**. The variable automatically points to the schema created in Step 7.

When complete, the Translate dialog looks as shown in Figure 6-12.

10. ***Figure 6-12 Translate Dialog Configured for Native to XML Translation***



The output for the synchronous request must now be changed to point to the new schema.

11. In the Applications window, select the BPEL process WSDL file (for this example, named **BPELProcess1.wsdl**).

12. At the bottom of Oracle BPEL Designer, click **Source**.

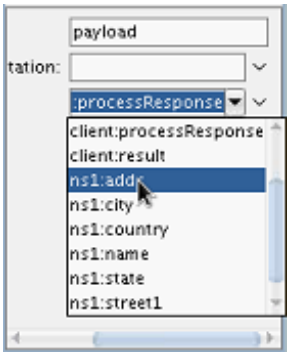
13. Scroll to the **<wsdl:message>** section of the WSDL file.

14. Click the response element (for this example, named **processResponse**) for the message **BPELProcess1ResponseMessage** to invoke the Property Inspector in the lower right corner. Figure 6-13 provides details.

Figure 6-13 Root Element Selection in the WSDL File

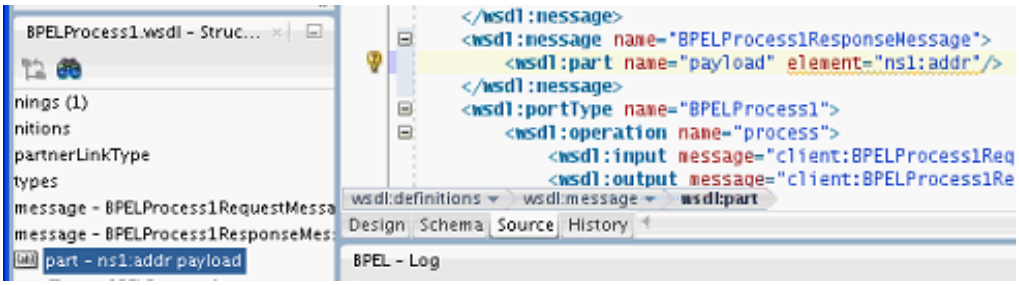
15. In the Property Inspector, select the new root element (for this example, **ns1:addr**). Figure 6-14 provides details.

Figure 6-14 Root Element Selected in Property Inspector



The **ns1:addr** root element is added to the WSDL file. Figure 6-15 provides details.

Figure 6-15 New Root Element Appears in WSDL File

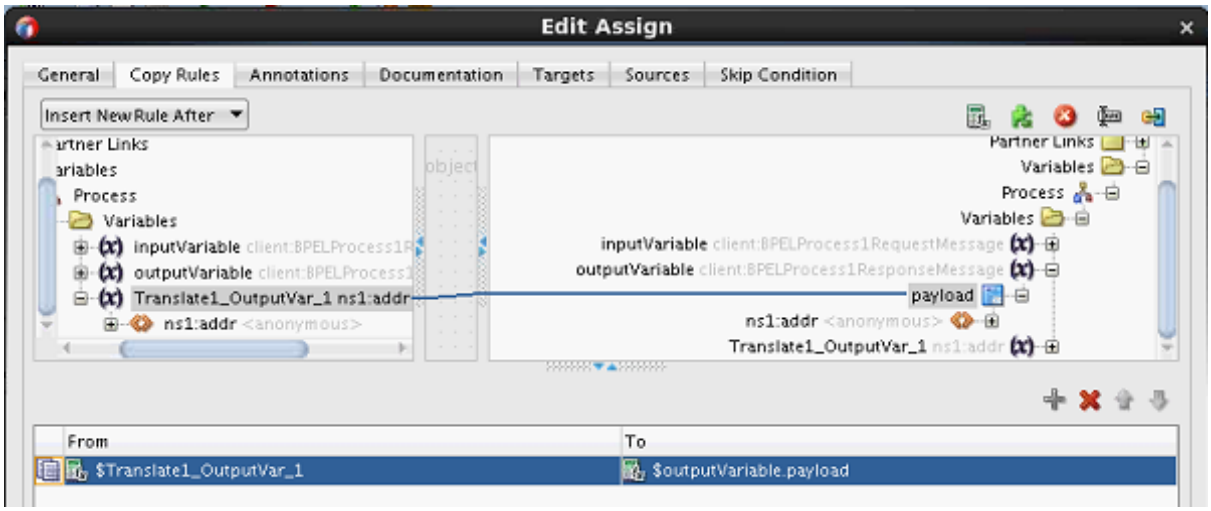


16. Drag an **Assign** activity into the BPEL process beneath the translate activity.

    You now assign the translation output variable to the BPEL output variable.

17. In the **Copy Rules** tab of the assign activity, map the variables, and click **OK**. Figure 6-16 provides details.

Figure 6-16 Edit Assign Dialog



Design is now complete.

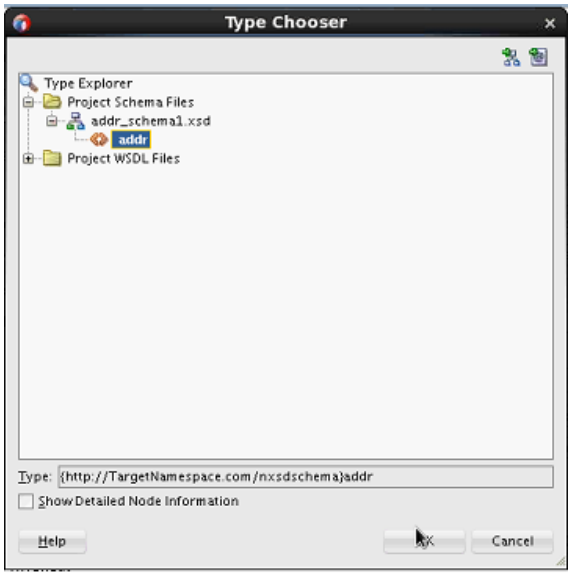
How to Translate XML Data to Native Data

This section describes how to translate an incoming XML message to native data format (such as a comma delimited string). This example uses the schema file created in How to Translate Native Data to XML Data as the outbound XML format to translate to native XSD format.

To translate XML format to native data:

1. Create a synchronous BPEL process.
2. In the **Input** field of the Create BPEL Process dialog, accept the default input XSD schema or click the **Search** icon to select a different XSD. For this example, the schema created with the Native Format Builder in How to Translate Native Data to XML Data is selected. Figure 6-17 provides details.

Figure 6-17 Input Schema Selection



3. Right-click the BPEL process in the SOA Composite Editor, and select **Edit**.

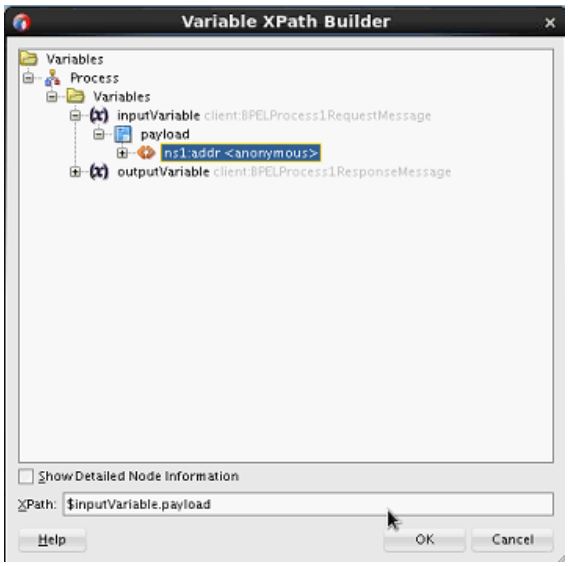
   Oracle BPEL Designer is displayed.

4. Expand the **Oracle Extensions** section of the Components window and drag a **Translate** activity into the BPEL process.
5. Right-click the translate activity, and select **Edit**.

   The Translate dialog is displayed for editing.

6. Select **XML to Native** to translate outbound XML data into native XSD format.
7. To the right of the **Input** field, click the **Browse** (first) icon.
8. Select the input variable. Figure 6-18 provides details.

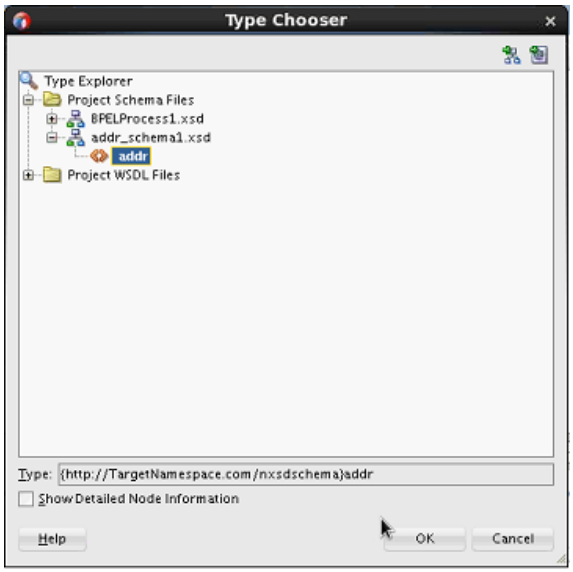
Figure 6-18 Input Variable Selection



9. To the right of the **NXSD Schema** field, select the **Search** (first) icon to invoke the Type Chooser dialog.

10. Select the schema file, and click **OK**. This example uses the same schema file as How to Translate Native Data to XML Data. Figure 6-19 provides details.

Figure 6-19 Schema File Selection



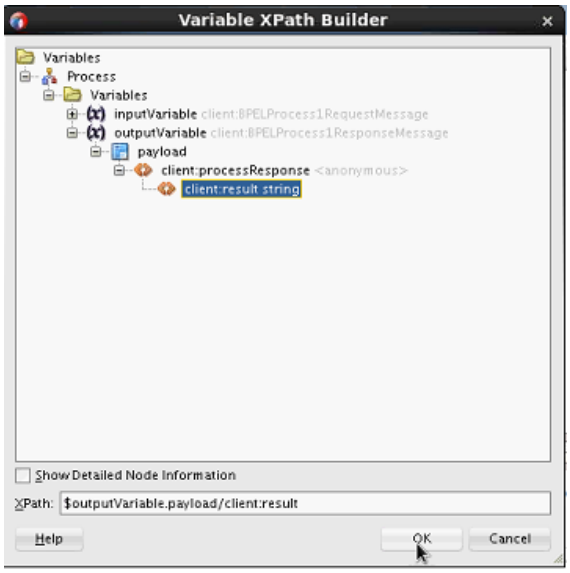
11. From the **Output Type** list, select **STRING**.

    If you instead select **ATTACHMENT**, the dialog is refreshed to display the **Location** field for specifying the directory location for the attachment. Selecting **ATTACHMENT** is useful for scenarios in which XML data is very large.

12. To the right of the **Output** field, click the **Search** (first) icon or click the **Create Variable** icon to automatically create a new output variable of type string.

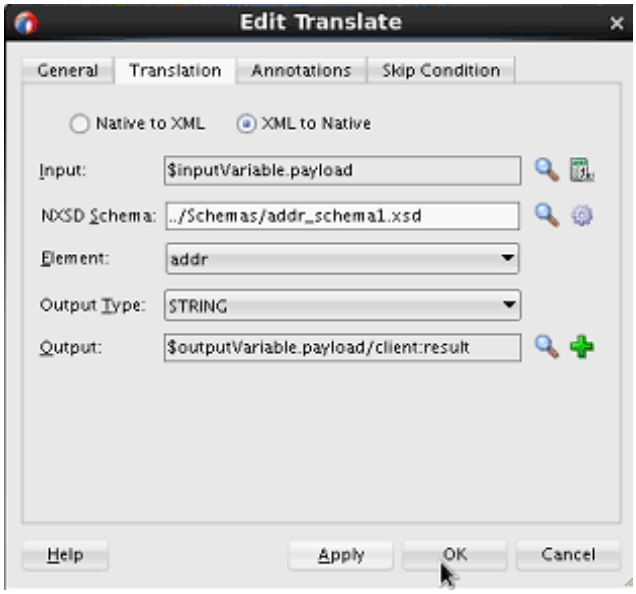
13. Select the output variable, and click **OK**. Figure 6-20 provides details.

Figure 6-20 Output Variable Selection



The Translate dialog looks as shown in Figure 6-21.

Figure 6-21 Translate Dialog Configured for Outbound Translations



Design is now complete.

How to Translate Inbound Native Data to XML Stored as an Attachment

This section describes how to translate an inbound message in native data format to an attachment. Attachments are useful for scenarios in which incoming data is very large.

To translate inbound native XSD format to an attachment:

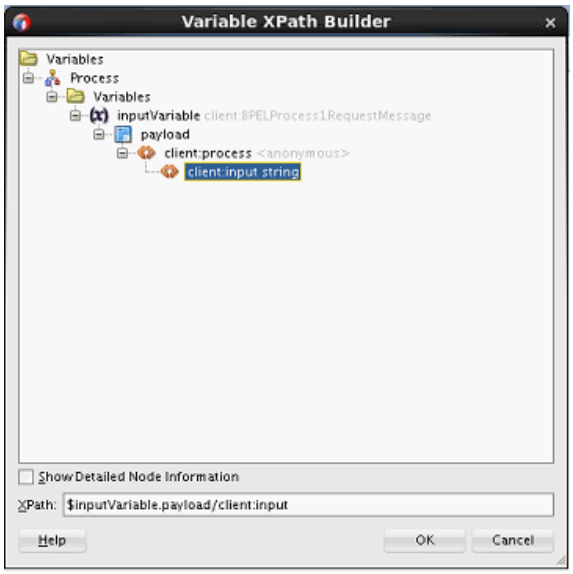
1. Create a BPEL process (for this example, a one-way BPEL process is created).
2. Right-click the BPEL process in the SOA Composite Editor, and select **Edit**.

   Oracle BPEL Designer is displayed.

3. Expand the **Oracle Extensions** section of the Components window and drag a **Translate** activity into the BPEL process.
4. Right-click the translate activity, and select **Edit**.

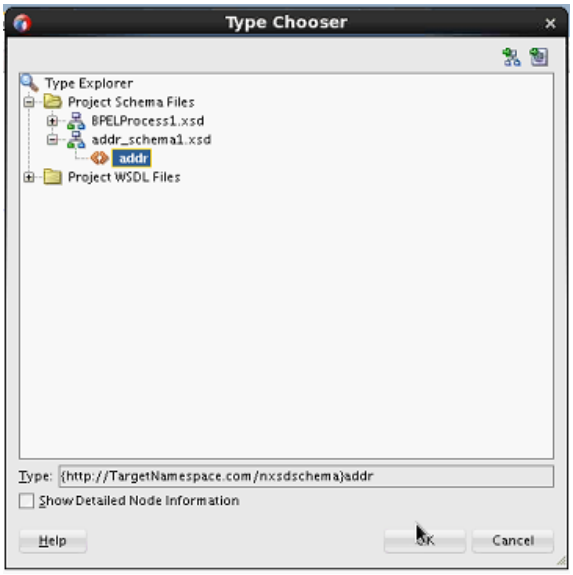
   The Translate dialog is displayed for editing.
5. Select **Native to XML** to translate inbound native data into an attachment.
6. To the right of the **Input** field, click the **Browse** (first) icon.
7. Select the input variable (for this example, a very large string). Figure 6-22 provides details.

Figure 6-22 Input Variable Selection



8. To the right of the **NXSD Schema** field, select the **Search** (first) icon to invoke the Type Chooser dialog.
9. Select the schema file, and click **OK**. This example uses the same schema file as How to Translate Native Data to XML Data. Figure 6-23 provides details.

Figure 6-23 Schema File Selection



10. From the **Output Type** list, select **ATTACHMENT**.

    The dialog is refreshed to display the **Location** field.

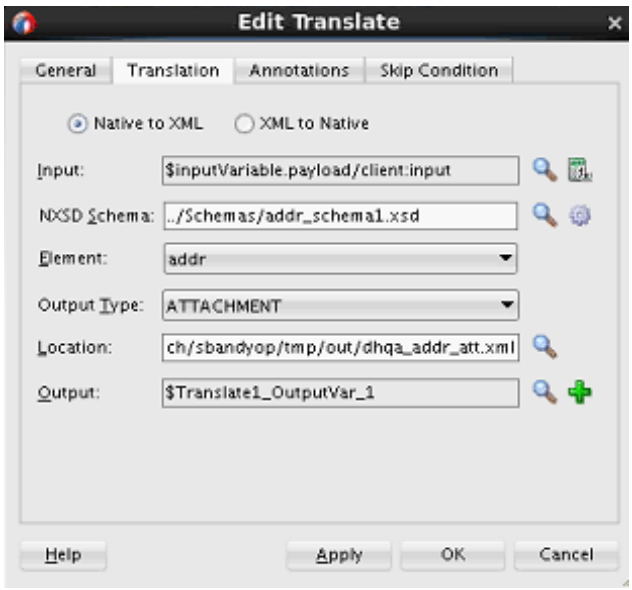
11. In the **Location** field, enter the directory path to the attachment. If this field is left blank, the attachment is stored in the database.

12. To the right of the **Output** field, click the **Add** (second) icon to invoke the Create Variable dialog.

13. Click **OK** to create the output variable. The output variable is of type attachment.

    The Translate dialog looks as shown in Figure 6-24.

Figure 6-24 Translate Dialog for an Attachment



14. Click **OK**.

15. In the Applications window, select the BPEL process file.
16. Click **Source**.
17. Note that the location you specified for the attachment is copied to an `href` attribute. The `href` attribute is part of the variable of type attachment that was created in Step 12.

```
. . .
<copy>
  <from> '/scratch/sbandyop/tmp/out/dhqa_addr_att.xml'</from>
  <to> $Translate1_OutputVar_1/@href</to>
</copy>
. . .
```

18. In the Applications window, select the BPEL process WSDL file.
19. Click **Source**.
20. Note the `attachment` code added to the WSDL definitions section of the file and the `href` attribute that is pointed to by the variable created in Step 12.

```
. . .
xmlns:attach="http://xmlns.oracle.com/DHQATranslateApp/DHQATranslateToAttach/
BPELProcess1/attachment"
. . .
. . .
. . .
   <element name="attachmentElement">
      <complexType>
         <attribute name="href" type="string"/>
      </complexType>
   </element>
. . .
. . .
```

Design is now complete.

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.

How to Declare SDO-based Variables

The syntax for declaring an SDO-based variable is similar to that for declaring BPEL variables. The following example provides details.

```
<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 the preceding sample is a regular DOM-based variable. The following example shows an 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>
```

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`).

The following example shows 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>
```

The following example illustrates copying 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>
```

The following example shows 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>` operation before the `<copy>` operation).
- The SDO data object is metadata driven and does not generally support adding a new property arbitrarily.

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. You assign literal XML by dragging a literal icon to a target node on the Copy Rules tab of the assign activity.

For more information about assigning literal XML in an assign activity, see Assign Activity.

How To Assign a Literal XML Element

The following example assigns a literal `result` element to the `payload` part of the `output` variable:

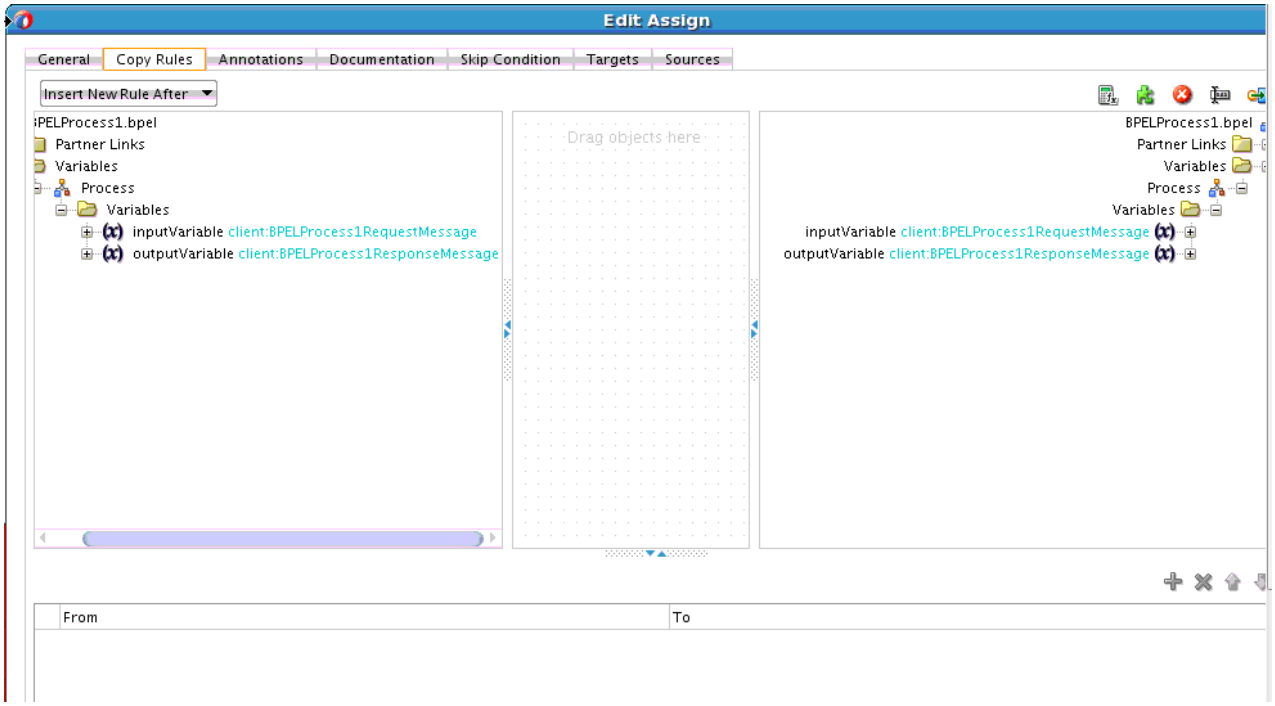
```
<assign>
   <!-- copy from literal xml to the variable -->
   <copy>
      <from>
         <result xmlns="http://samples.otn.com">
            <name/>
            <symbol/>
            <price>12.3</price>
            <quantity>0</quantity>
            <approved/>
            <message/>
         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

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.

You perform variable copying in the **Copy Rules** tab of the Edit Assign dialog, as shown in Figure 6-25.

Figure 6-25 Copy Rules Tab for Variable Assignment



For more information about the **Copy Rules** tab, see Manipulating XML Data with bpelx Extensions and Assign Activity.

How to Copy Between Variables

The following example 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.

```
<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, as shown in the following example:

```
<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, as shown in the following example:

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

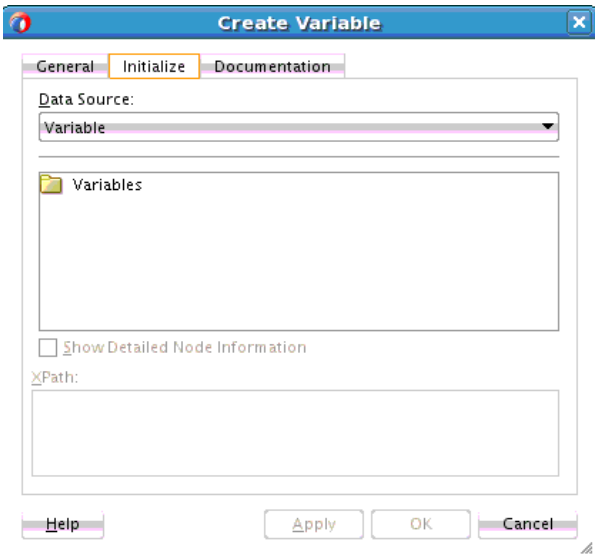
For more information about this code example, see Section 9.3.2 of the . For BPEL 2.0, see Section 8.4.4 of for a similar example.

For more information, see Assign Activity.

How to Initialize 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. Figure 6-26 provides details.

Figure 6-26 Initialize Tab of Create Variable Dialog



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. The following example provides details.

```
<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 .

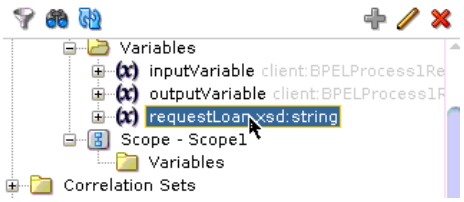
Moving and Copying Variables in the Structure Window

You can move and copy variables to and from scope activities in the Structure Window of Oracle JDeveloper.

To Move Variables in the Structure Window:

1. In the Structure window, select the variable to move to a scope activity. Figure 6-27 provides details.

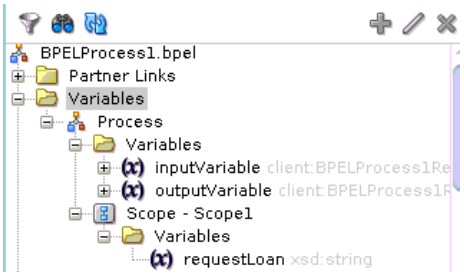
Figure 6-27 Variable to Move in the Structure Window



2. Drag the variable to the **Variables** folder of the scope activity.

   The variable is displayed in the **Variables** folder of the scope activity, as shown in Figure 6-28.

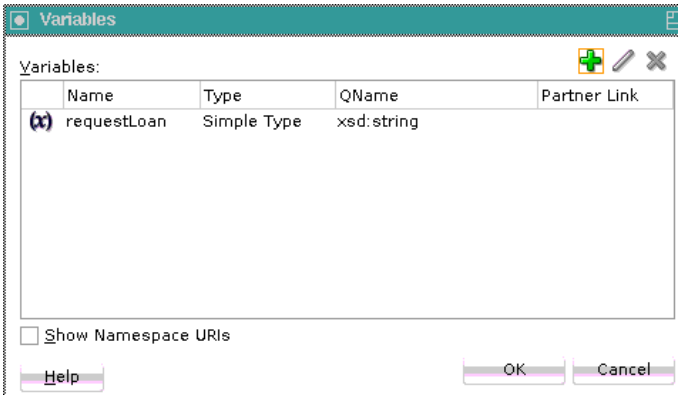
Figure 6-28 Variable Moved to the Scope Activity in the Structure Window



3. In the BPEL process, click the **Variables** icon of the scope activity.

   The variable you moved is displayed, as shown in Figure 6-29.

Figure 6-29 Moved Variable in Variables Dialog of the Scope Activity

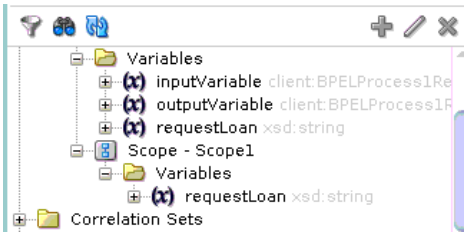


To Copy Variables in the Structure Window:

1. In the Structure window, select the variable to move to the scope activity.
2. Hold down the Ctrl key.
3. Drag the variable to the **Variables** folder of the scope activity.

The variable is displayed in both **Variables** folders, as shown in Figure 6-30.

Figure 6-30 Variable Copied to the Scope Activity in the Structure Window



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 **Copy Rules** tab of the assign activity.

For more information about the **Copy Rules** tab, see Manipulating XML Data with bpelx Extensions and Assign Activity.

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

In the following example, the `ssn` field is copied from the `CreditFlow` process's input message into the `ssn` field of the credit rating service's input message.

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

The following example shows how the BPEL file defines message type-based variables involved in this assignment:

```
<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 the following example:

```
<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 the following example:

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

The following example shows the BPEL 2.0 syntax for how the BPEL file defines message type-based variables involved in the assignment in the first assignment example. Note that `/tns:CreditFlowRequest` is not required.

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

A BPEL process can also use element-based variables. The following example 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.

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

The following example shows how to use element-based variables in BPEL 2.0.

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

The following example shows how the BPEL file defines element-based variables involved in an assignment:

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

Assigning Numeric Values

You can assign numeric values in XPath expressions.

How to Assign Numeric Values

The following example shows how to assign an XPath expression with the integer value of `100`.

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

Using Mathematical Calculations with XPath Standards

You can use simple mathematical expressions, such as the one in the following section, which increment a numeric value.

How To Use Mathematical Calculations with XPath Standards

In the following example, 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).

```
<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 the following example:

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

The following example shows how to use `$variable` syntax in BPEL 2.0.

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

Assigning String Literals

You can assign string literals to a variable in the **Copy Rules** tab of the assign activity.

For more information about the assign activity, see Manipulating XML Data with bpelx Extensions and Assign Activity.

How to Assign String Literals

The code in the following example 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.)

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

The following example shows how to perform this expression in BPEL 2.0.

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

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

For more information, see Assign Activity.

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.

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 the following example, `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.

```
<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 .

Assigning Boolean Values

You can assign boolean values with the XPath boolean function.

How to Assign Boolean Values

The following example 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.

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

The following example provides an example of assigning boolean values 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.

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 and section 8.3.2 of the .

For information about XPath functions and the Expression Builder, see XPath Extension Functions.

How to Assign a Date or Time

The following example shows an example that uses the function getCurrentDate in BPEL 1.1.

```
<!-- execute the XPath extension function getCurrentDate() -->
<assign>
   <copy>
      <from expression="xpath20:getCurrentDate()"/>
      <to variable="output" part="payload"
         query="ns1:invoice/invoiceDate"/>
   </copy>
</assign>
```

The following example shows an example that uses the function getCurrentDate in BPEL 2.0.

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

In the following example, 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).

```
<!-- 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="ns1:invoice/formattedDate"/>
   </copy>
</assign>
```

The following example shows how the formatDate function works 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>
```

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.

How to Manipulate Attributes

The code in the following example fetches and copies the `custId` attribute from this XML data:

```
<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 the following example selects the `custId` attribute of the customer field and assigns it to the variable `custId`:

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

The following example shows the equivalent syntax in BPEL 2.0 for selecting the `custId` attribute of the customer field and assigning it to the variable `custId`:

```
<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 the following example:

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

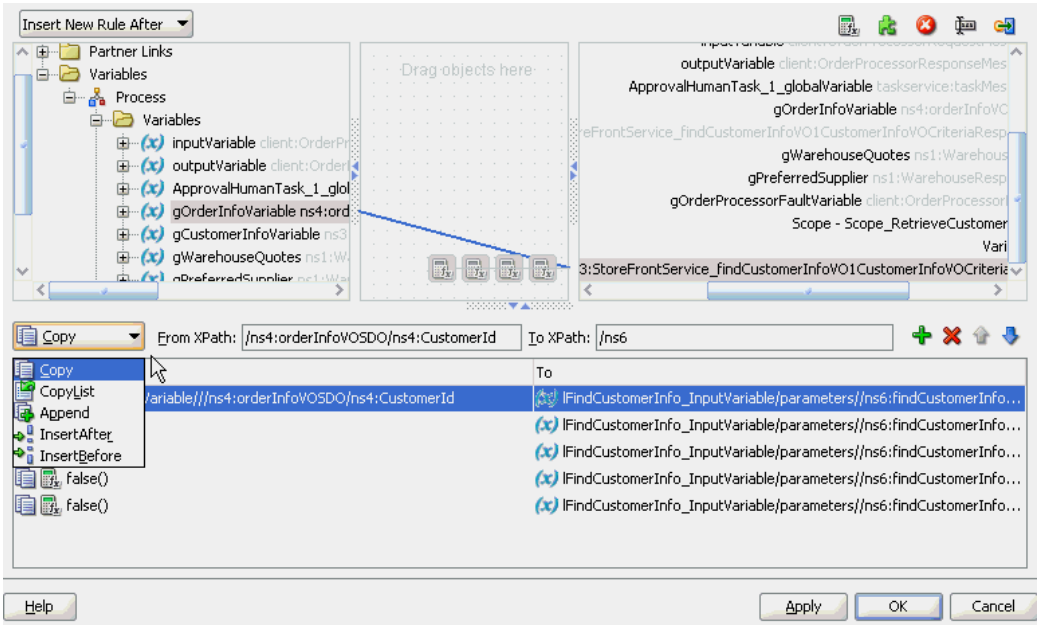
```
   <attribute name="custId" type="string"/>
</complexType>
```

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.

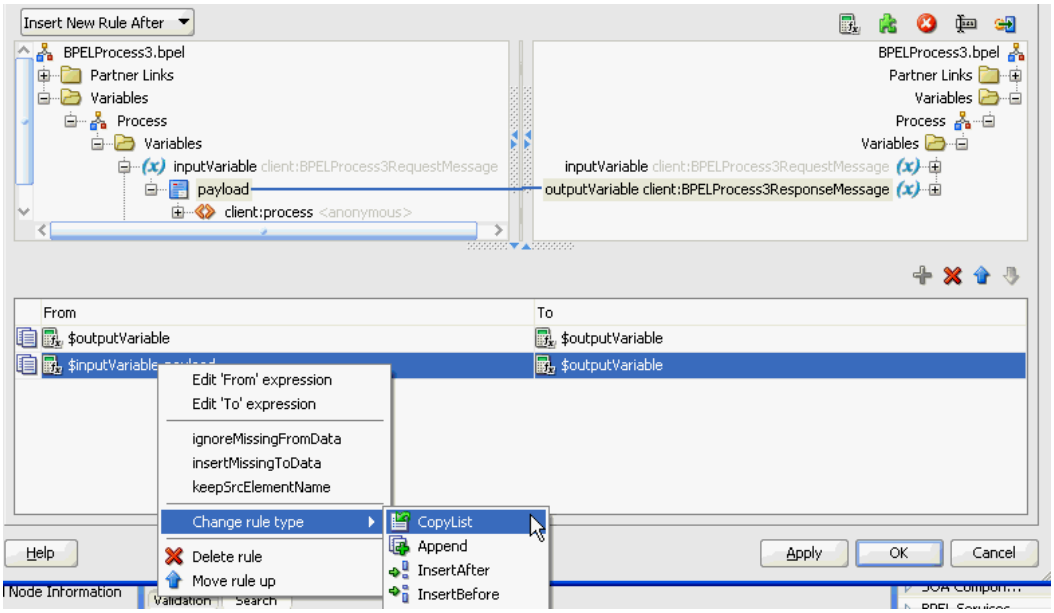
In BPEL 1.1, you select an extension type from the dropdown list, as shown in Figure 6-31.

Figure 6-31 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 6-32.

Figure 6-32 Copy Rule Converted to bpelx Extension in BPEL 2.0



For more information, see the online Help for this dialog and Assign Activity.

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, perform one of the following steps at the bottom of the **Copy Rules** tab:

- For BPEL 1.1, select a copy rule, then select **Append** from the dropdown list, as shown in Figure 6-31.
- For BPEL 2.0, right-click a copy rule, select **Change rule type**, and then select **Append**, as shown in Figure 6-32.

Note:

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

---

bpelx:append in BPEL 1.1

The following provides an example of `bpelx:append` in a BPEL project that supports BPEL version 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.

The following example 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.

```
<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>
```

bpelx:append in BPEL 2.0

The following 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 bpelx:append in BPEL 1.1, but the syntax is slightly different.

```
<bpel:assign>
   <bpelx:append>
      <bpelx:from>$billOfMaterialVar/b:parts/b:part</bpelx:from>
      <bpelx:to>$consolidatedBillOfMaterialVar/b:parts</bpelx:from>
   </bpelx:append>
</bpel:assign>
```

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, perform one of the following steps at the bottom of the **Copy Rules** tab:

- For BPEL 1.1, select a copy rule, then select **InsertBefore** from the dropdown list, as shown in Figure 6-31.
- For BPEL 2.0, right-click a copy rule, select **Change rule type**, and then select **InsertBefore**, as shown in Figure 6-32.

bpelx:insertBefore in BPEL 1.1

The following provides an example of `bpelx:insertBefore` in a BPEL project that supports BPEL version 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.

The following example shows the syntax before the execution of `<insertBefore>`. The value of `addrVar` is:

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

The following example shows the syntax after the 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>
```

The following example shows the value of `addrVar`:

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

bpelx:insertBefore in BPEL 2.0

The following 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 bpelx:insertBefore in BPEL 1.1, but the syntax is slightly different. An `extensionAssignOperation` element wraps the `bpelx:insertBefore` extension.

```
<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>
```

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, perform one of the following steps at the bottom of the **Copy Rules** tab:

- For BPEL 1.1, select a copy rule, then select **InsertAfter** from the dropdown list, as shown in Figure 6-31.
- For BPEL 2.0, right-click a copy rule, select **Change rule type**, and then select **InsertAfter**, as shown in Figure 6-32.

bpelx:insertAfter in BPEL 1.1

The following provides an example of `bpelx:insertAfter` in a BPEL project that supports BPEL version 1.1.

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

This operation is similar to the functionality described for 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`).

The following example shows the syntax before the execution of `<insertAfter>`. The value of `addrVar` is:

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

The following example shows the syntax after the 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>
```

The following example shows the value of `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>
```

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.

bpelx:insertAfter in BPEL 2.0

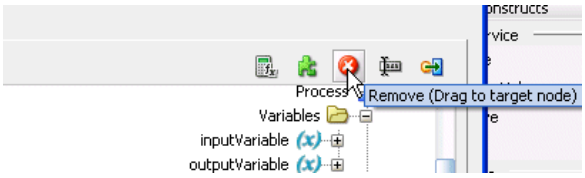
The following provides an example of `bpelx:insertAfter` syntax in a BPEL project that supports BPEL version 2.0. In BPEL 2.0, the functionality is the same as described in bpelx:insertAfter in BPEL 1.1, but the syntax is slightly different. An `extensionAssignOperation` element wraps the `bpelx:insertAfter` extension.

```
<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>
```

How to Use bpelx:remove

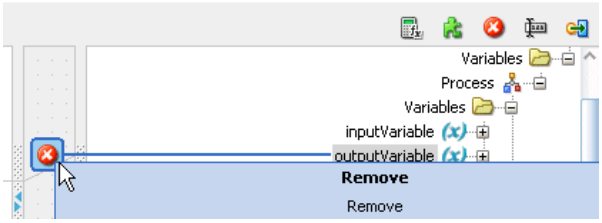
The `bpelx:remove` extension in an assign activity enables a BPEL process service component to remove a variable. In Oracle BPEL Designer, you add the `bpelx:remove` extension by dragging the **remove** icon in the upper right corner of the **Copy Rules** tab to the target variable you want to remove, and releasing the cursor. You can also drag this icon to the center canvas to invoke a dialog, specify the rule, save and close the dialog, and then drag the icon to the target node. Figure 6-33 provides details.

Figure 6-33 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 6-34 provides details.

Figure 6-34 bpelx:remove Extension Applied to a Target Variable



bpelx:remove in BPEL 1.1

The following provides an example of `bpelx:remove` in a BPEL project that supports BPEL version 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.

The following example shows `addrVar` with the following 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>
```

After executing the syntax shown in the BPEL process service component file, the second address line of `Mailstop` is removed:

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

After executing the syntax shown in the BPEL process service component file, both address lines are removed:

```
<bpel:assign>
    <bpelx:remove>
        <target variable="addrVar"
            query="/a:usAddress/a:addressLine" />
    </bpelx:remove>
</bpel:assign>
```

bpelx:remove in BPEL 2.0

The following 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 bpelx:remove in BPEL 1.1, but the syntax is slightly different. An `extensionAssignOperation` element wraps the `bpelx:remove`.

```
<assign>
   <extensionAssignOperation>
      <bpelx:remove>
         <bpelx:target>$ncname.ncname/xpath_str</bpelx:target>
      </bpelx:remove>
   </extensionAssignOperation>
</assign>
```

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 rename, and releasing the cursor. The rename icon displays to the right of the **remove** icon shown in Figure 6-33. After releasing the cursor, the Rename dialog is displayed for renaming the target variable. You can also drag this icon to the center canvas to invoke this dialog, specify the name, save and close the dialog, and then drag the icon to the target node.

bpelx:rename in BPEL 1.1

The following provides an example of `bpelx:rename` in a BPEL project that supports BPEL version 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 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 the following example:

```
<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, as in the following example:

```
<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 the following example with `xsi:type` info added to `Peter Smith`:

```
<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. The following provides an example.

```
<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 the following example:

```
<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>
```

bpelx:rename in BPEL 2.0

The following 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 bpelx:rename in BPEL 1.1, but the syntax is slightly different. An `extensionAssignOperation` element wraps the `bpelx:rename` operation.

```
 <bpel:assign>
   <extensionAssignOperation>
      <bpelx:rename elementTo="QName1"? typeCastTo="QName2"?>
         <bpelx:target>$ncname[.ncname][/xpath_str]</bpelx:target>
      </bpelx:rename>
   </extensionAssignOperation>
</bpel:assign>
```

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, perform one of the following steps at the bottom of the **Copy Rules** tab:

- For BPEL 1.1, select a copy rule, then select **CopyList** from the dropdown list, as shown in Figure 6-31.
- For BPEL 2.0, right-click a copy rule, select **Change rule type**, and then select **CopyList**, as shown in Figure 6-32.

bpelx:copyList in BPEL 1.1

The following provides an example of `bpelx:copyList` in a BPEL project that supports BPEL version 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 below:

```
<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 the following example:

```
<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 the following example:

```
<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 the following example:

```
<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 defines the `to` variable as shown in the following example:

```
<ns1:processResponse xmlns:ns1="http://xmlns.oracle.com/Event_
 jws/Event/EventTest">
           <ns1: payload >a</ns1: payload >
           <ns1: payload >b</ns1: payload >
</ns1:process>
```

bpelx:copyList in BPEL 2.0

The following 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 bpelx:copyList in BPEL 1.1, but the syntax is slightly different. An `extensionAssignOperation` element wraps the `bpelx:copyList` extension.

```
<assign>
   <extensionAssignOperation>
      <bpelx:copyList>
         <bpelx:from>$inputVariable.payload/client:payload</bpelx:from>
         <bpelx:to>$outputVariable.payload/client:payload</bpelx:to>
      </bpelx:copyList>
   </extensionAssignOperation>
</assign>
```

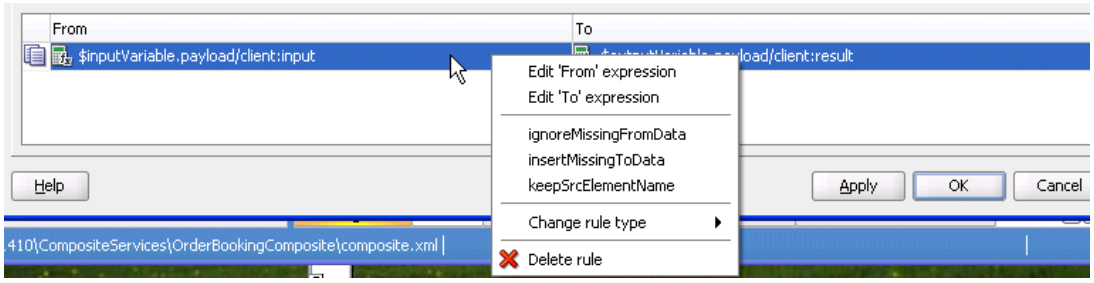
How to Use Assign Extension Attributes

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 6-35 provides details.

Figure 6-35 Assign Extension Attributes



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"/>`

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"/>`

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"/>`

Validating XML Data

You can verify code and identify invalid XML data in a BPEL project.

How to Validate XML Data in BPEL 2.0

This section discusses validating XML data in BPEL 2.0.

Validate XML in an Assign Activity

In an assign activity in Oracle BPEL Designer:

1. From the **BPEL Constructs** section of the Components window, 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** check box.
4. Click **Apply**, then **OK**.

5. Click the **Source** tab to view the syntax. 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>
```

Validate XML in a Standalone, Extended Validate Activity

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 Components window, 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. 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"/>
```

How to Validate XML Data in BPEL 1.1

This section describes validating xml data in BPEL 1.1.

Validate XML in an Assign Activity

In an assign activity in Oracle BPEL Designer:

1. From the **BPEL Constructs** section of the Components window, 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** check box.
4. Click **Apply**, then **OK**.
5. Click the **Source** tab to view the syntax.

```
<assign name=Assign1" bpelx:validate="yes"
  . . .
</assign>
```

Validate XML in a Standalone, Extended Validate Activity

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 Components window, 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"/>
```

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 activity 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 keepSrcElementName Attribute.

Mapping WSDL Message Parts in BPEL 2.0

The **Arguments Mapping** section in invoke and reply activities provides an alternative to explicitly creating multipart WSDL messages from the contents of BPEL variables.

When you use the **Arguments Mapping** section, an anonymous, temporary WSDL variable is defined based on the type specified by the input message of the appropriate WSDL operation.

For more information about mapping WSDL message parts, see the located at the following URL:

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

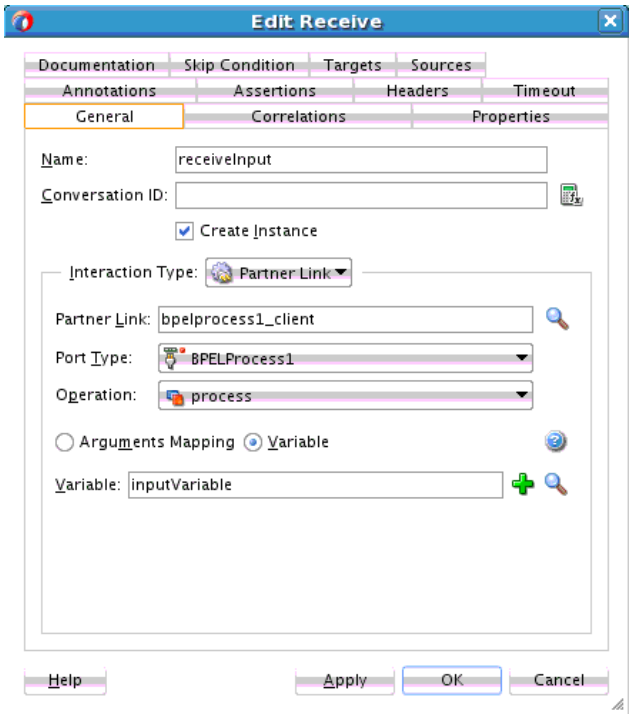
How to Map WSDL Message Parts

The **Arguments Mapping** table contains the parts for the selected operation. You can set the value of each message part by editing the **Value** column of the table. Select the variable in which to retrieve the value and store the message part.

To map WSDL message parts in BPEL 2.0:

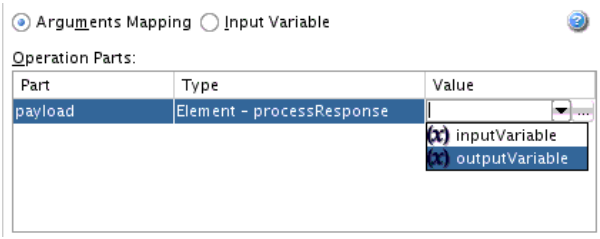
1. Note that the receive activity in Figure 6-36 includes a standard **inputVariable** variable from the client.

Figure 6-36 Receive Activity



2. Note the **Arguments Mapping** button at the bottom of the reply activity in Figure 6-37. You can set the value for each message part by clicking an entry in the table.

Figure 6-37 Arguments Mapping Section Defined at Bottom of a Reply Activity



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 the following example:

```
<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 the following example:

```
<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 the following example:

```
<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 .

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.

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 the following example, `addresses[1]` selects the first element of the addresses data sequence:

```
<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). The query in the following example calls the `position` function explicitly to select the first element of the address's data sequence. It then selects that address's `street` element (which the activity assigns to the variable `street1`).

```
<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.

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.

The following example provides an example of a SOAP array payload named `myFavoriteNumbers`.

```
<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. The following example provides details:

```
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:

The following example 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` is used to add items into the 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/" />
```

SOAP-Encoded Arrays in BPEL 2.0

SOAP-encoded arrays are supported in BPEL projects that use version 2.0 of the BPEL specification. The following example shows a sample assign activity with a SOAP-encoded array in a BPEL 2.0 project.

```
<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>
```

The following example shows a sample invoke activity with a SOAP-encoded array in a BPEL 2.0 project.

```
<invoke name="Invoke1" partnerLink="FileOut"
   portType="ns3:Write_ptt" operation="Write"
   bpelx:invokeAsDetail="no">
   <toParts>
      <toPart part="body" fromVariable="ArrayVariable"/>
   </toParts>
</invoke>
```

Declaring a SOAP Array Using a wsdl:arrayType Attribute Inside a Schema

A SOAP-encoded array WSDL can declare a SOAP array using a `wsdl:arrayType` attribute inside a schema. The following example provides details.

```
<xsd:complexType name="UserObject">
    <xsd:sequence>
        <xsd:element name="userInformation" nillable="true"
          type="n5:ArrayOfKeyValuePair"/>
        <xsd:element name="username" nillable="true" type="xsd:string"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="ArrayOfKeyValuePair">
    <xsd:complexContent>
        <xsd:restriction base="soapenc:Array">
            <xsd:attribute ref="soapenc:arrayType"
             wsdl:arrayType="n5:KeyValuePair[]"/>
        </xsd:restriction>
```

```
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="KeyValuePair">
    <xsd:sequence>
        <xsd:element name="key" nillable="true" type="xsd:string"/>
        <xsd:element name="value" nillable="true" type="xsd:string"/>
    </xsd:sequence>
</xsd:complexType>
```

The following example shows how to create and access a SOAP-encoded array in BPEL 1.1.

```
<bpws:copy>
    <bpws:from>
       <ns1:userInformation soapenc:arrayType="com1:KeyValuePair[1]"
       xmlns:ns1="http://www.schematargetnamespace.com/wsdl/Impl/"
         xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"/>
         <ns1:KeyValuePair
            xmlns:ns1="http://www.schematargetnamespace.com/wsdl/Impl/">
            <key>testkey</key>
            <value>testval1</value>
          </ns1:KeyValuePair>
       </ns1:userInformation>
    </bpws:from>
    <bpws:to variable="Inputvar" part="userObject"
             query="/userObject/userInformation"/>

</bpws:copy>
<!--Update elements with SOAPENC Array-->
<bpws:copy>
    <bpws:from variable="KeyValueVar" part="KeyValuePair"
                query="/KeyValuePair/ns2:key"/>
    <bpws:to variable="Inputvar" part="userObject'
 query="//*[local-name()='KeyValuePair'][1]/*[local-name()='key']"/>
</bpws:copy>

<bpws:copy>
    <bpws:from variable="KeyValueVar" part="KeyValuePair"
                query="/KeyValuePair/client:value"/>
    <bpws:to variable="Inputvar" part="userObject"
   query="//*[local-name()='KeyValuePair'][1]/*[local-name()='value']"/>

</bpws:copy>
<!-- Append elements within SOAPENC Array -->
<bpelx:append>
    <bpelx:from variable="Inputvar" part="userObject"
                query="//*[local-name()='KeyValuePair'][1]"/>
    <bpelx:to variable="Inputvar" part="userObject"
              query="/userObject/userInformation"/>
</bpelx:append>
```

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 the following example calculates the number of elements in the `item` sequence and assigns it to the integer variable `lineItemSize`.

```
<assign>
   <copy>
      <from expression="count(bpws:getVariableData('outpoint', 'payload',
                        '/p:invoice/p:lineItems/p:item')"/>
      <to variable="lineItemSize"/>
   </copy>
</assign>
```

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.

Applying a Trailing XPath to the Result of getVariableData

The dynamic indexing method shown in the following example 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 makes reference to an integer-based index variable within the position predicate (that is, `[...]`).

```
<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 expression in the preceding example within the `from` is equivalent to that shown in the following example.

```
<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.

The following example provides details.

```
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.

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. The following provides an example.

```
<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.

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 below under `assign` demonstrate how to merge data sequences:

```
<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>
```

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.

The following example shows an `append` statement initializing a purchase order (PO) document with `10` empty `<lineItem>` elements under `po`:

```
<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 the previous example can be replaced with an embedded XQuery expression, as shown in the following example:

```
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.

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. The following example provides details.

```
<?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>
```

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 parseEscapedXML.

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. The following provides an example:

```
<!-- 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>
```

Understanding Document-Style and RPC-Style WSDL Differences

The examples provided in previous sections of this chapter have been for document-style WSDL files in which a message is defined with an XML schema `element`, as shown in he following example:

```
<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 the following example:

```
<message name="LoanFlowRequestMessage">
<part name="payload" type="s1:LoanApplicationType"/>
</message>
```

How To Use RPC-Style Files

This differs from the previous information in this chapter because there is a difference in how XPath queries are constructed for the two WSDL message styles. For an RPC-

style message, the top-level element (and therefore the first node in an XPath query string) is the part name (`payload` in the previous example). In document-style messages, the top-level node is the element name (for example, `loanApplication`).

The following examples (WSDL file and BPEL file) show what an XPath query string looks like if an application named `LoanServices` were in RPC style.

```
<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>

<variable name="output"
          messageType="tns:LoanServiceResultMessage"/>
...
<assign>
   <copy>
      <from expression="9.9"/>
      <to variable="output" part="payload" query="/payload/APR"/>
   </copy>
</assign>
```

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 the following example:

```
<invoke bpelx:inputHeaderVariable="inHeader1 inHeader2 ..."
  bpelx:outputHeaderVariable="outHeader1 outHeader2 ..."
  .../>

<receive bpelx:headerVariable="inHeader1 inHeader2 ..." .../>
<onMessage bpelx:headerVariable="inHeader1 inHeader2 ..." .../>
<reply bpelx:headerVariable="inHeader1 inHeader2 ..." .../>
```

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. The following provides an example:

```
<!-- 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 the following example:

```
<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"/>
```

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 a reference in the `composite.xml` file to refer to the `HeaderService`.

2. Define the custom header variable, manipulate it, and send it using `bpelx:inputHeaderVariable`, as shown in the following example:

```
<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"/>
```

Declaring Extension Namespaces in BPEL 2.0

You can extend a BPEL version 2.0 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. The following example provides details.

```
<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 located at the following URL:

http://www.oasis-open.org

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** check box.

5. Click **OK**.
6. Click **Close**.

What Happens When You Create an Extension

After you complete your design, the `.bpel` process looks as shown in the following example:

```
<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. It demonstrates how to set up the components necessary to perform a synchronous invocation and how these components are coded. It also describes how to specify a timeout value and call a one-way Oracle Mediator with a synchronous BPEL process.

This chapter includes the following sections:

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 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 **SOAP** service from the **Technology** section of the Components window into the **Exposed Services** or **External References** swimlane. For more information, see Adding Service Binding Components or Adding Reference Binding Components.

  – In Oracle BPEL Designer, when you drag a **Partner Link** icon from the **BPEL Constructs** section of the Components window into the **Partner Links** swimlane. This second 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.

Invoking a Synchronous Web Service

This section examines a synchronous invocation operation using a file named `OrderProcessor.bpel`.

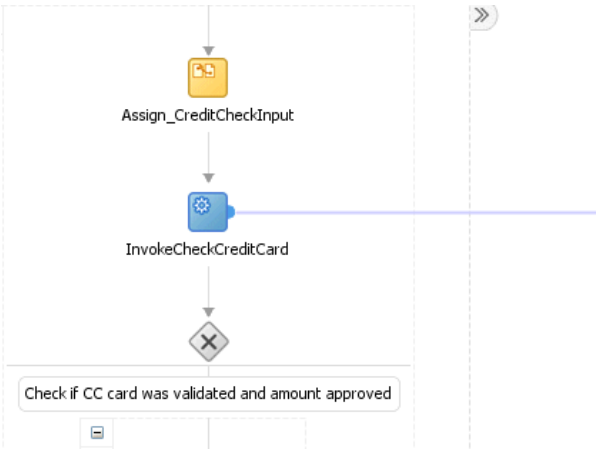
How to Invoke a Synchronous Web Service

To invoke a synchronous web service:

1. In the Components window 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 a scope activity named **Scope_AuthorizeCreditCard** of a BPEL process named **OrderProcessor**, which defines a simple set of actions.

Figure 7-1 Diagram of OrderProcessor.bpel



How Does the BPEL Process Work

The following actions take place:

1. The **Assign_CreditCardCheckInput** 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 **CreditCardAuthorizationService** service.
2. The **InvokeCheckCreditCard** invoke activity calls the **CreditCardAuthorizationService** service. Figure 7-2 shows the **CreditCardAuthorizationService** web service, which is defined as a partner link.

Figure 7-2 CreditCardAuthorizationService Partner Link

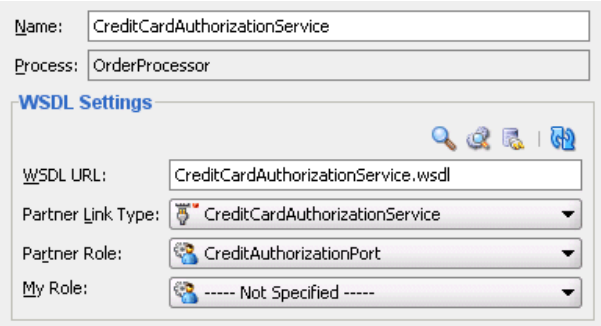
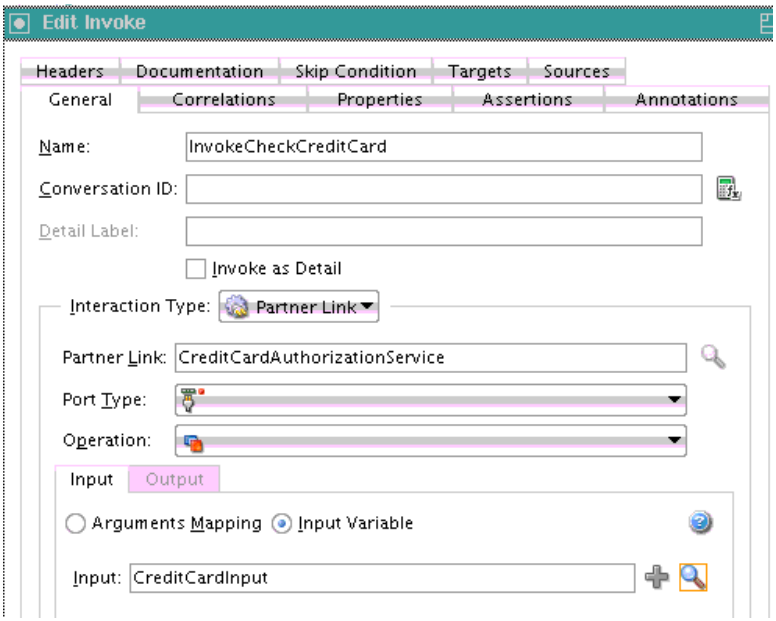


Figure 7-3 shows the **InvokeCheckCreditCard** invoke activity.

Figure 7-3 InvokeCheckCreditCard Invoke Activity



3. An if activity (for BPEL 2.0) or a switch activity (for BPEL 1.1) checks the results of the credit card validation. For information about if and switch activities, see Defining Conditional Branching with the If or Switch Activity.

Note:

The BPEL 2.0 if activity replaces the BPEL 1.1 switch activity.

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.

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 below:

```
<partnerLink name="CreditCardAuthorizationService"
    partnerRole="CreditAuthorizationPort"
    partnerLinkType="ns2:CreditCardAuthorizationService"/>
```

Variable definitions that are accessible locally in the `Scope_AuthorizeCreditCard` scope are shown in the following example. The types for these variables are defined in the WSDL for the process itself.

```
<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
- Other parameters

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

  – 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.

  The following provides an example of `partnerLinkType` and `portType`.

  ```
  <plnk:partnerLinkType name="CreditCardAuthorizationService">
       <plnk:role name="CreditAuthorizationPort">
           <plnk:portType name="tns:CreditAuthorizationPort"/>
       </plnk:role>
  </plnk:partnerLinkType>
  ```

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 `CreditCardAuthorizationService` service. The following example provides details.

```
<invoke name="InvokeCheckCreditCard"
    inputVariable="lCreditCardInput"
    outputVariable="lCreditCardOutput"
    partnerLink="CreditCardAuthorizationService"
    portType="ns2:CreditAuthorizationPort"
    operation="AuthorizeCredit"/>
```

Synchronous Invocation in BPEL Code

The BPEL code shown in the following example performs the 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"/>
```

Specifying Transaction Timeout Values in Durable Synchronous Processes

You can specify transaction timeout values with the property **SyncMaxWaitTime** in the System MBean Browser of Oracle Enterprise Manager Fusion Middleware Control. The **SyncMaxWaitTime** property applies to durable synchronous processes that are called in an asynchronous manner. If the BPEL process service component does not receive a reply within the specified time, then the activity fails. For more information, see What You May Need to Know About SyncMaxWaitTime and Durable Synchronous Requests Not Timing Out.

How To Specify Transaction Timeout Values

To specify transaction 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**.

What You May Need to Know About SyncMaxWaitTime and Durable Synchronous Requests Not Timing Out

The **SyncMaxWaitTime** property applies to durable synchronous processes that are called in an asynchronous manner.

Assume you have a BPEL process with the definition shown in the following example. The process is not durable because there are 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 the following example:

```
<receive name="receiveInput" partnerLink="client" variable="input"
createInstance="yes" />
<assign>
...
</assign>
<wait name="Wait1">
      <for>'PT10S'</for>
</wait>
<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.

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 `bpel.config.transaction` property to `requiresNew` in the `composite.xml` file. The following example provides details.

```
<component name="BPELProcess1">
  <implementation.bpel src="BPELProcess1.bpel"/>
  <property name="bpel.config.transaction" type="xs:string"
  many="false">requiresNew</property>
  </component>
```

The `requiresNew` value is recommended. If you want to participate in the client's transaction, you must set the `bpel.config.transaction` property to `required`.

8

Invoking an Asynchronous Web Service from a BPEL Process

This chapter describes how to configure and invoke an asynchronous web service from a BPEL process. It also describes how to route callback messages to the correct endpoint when multiple receive or pick activities use the same partner link, manage idempotence at the partner link operation level, create a dynamic partner link at runtime, override security certificates and WSDL files in dynamic partner link environments, and use WS-Addressing.

This chapter includes the following sections:

Introduction to Invoking 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 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.

Subsequent sections in this chapter provide specific details about the asynchronous functionality.

Invoking an Asynchronous Web Service

This section provides an overview of the tasks for adding asynchronous functionality to a BPEL process service component.

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

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 **Components** section of the Components window into the designer.

   The Create BPEL Process dialog appears.
2. Follow the instructions in the dialog to create an asynchronous 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**).

Oracle BPEL Designer appears.

5. In the Components window, 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 Browser** 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.

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 Components window, expand **BPEL Constructs**.
2. Drag an **Invoke** activity to beneath the **Receive** activity.

3. Go to the Structure window. 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, **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.
7. Click **OK**.
8. Click the **invoke** activity to display its property fields in the Property Inspector or double-click the **invoke** activity to display the Invoke dialog.

   For information about editing activities in the Property Inspector, see How to Edit BPEL Activities in the Property Inspector.
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 Invoke and Receive Activities.
11. Click **OK**.

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 Components window, drag a **Receive** activity to the location right after the **Invoke** activity you created in 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 Adding an Invoke Activity.

   Note:

   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). BPEL 2.0 is the default version when you create a BPEL process. For more information, see How to Initialize 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** check box.
6. Select the variable you created in Step 3 through Step 7 of 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.

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.

What Happens When You Invoke an Asynchronous Web Service

This section describes what happens when you invoke an asynchronous web service.

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
- The other calls back the client with the asynchronous response

In the example shown below, the `portType LoanServiceCallback` receives the client's loan application request and the `portType LoanService` asynchronously calls back the client with the loan offer response.

```
<!-- 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>
```

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 the following example, 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.

```
<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.

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. The following provides an example.

```
<!-- 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 `myRole` and `partnerRole` attributes in asynchronous processes.

Composite Application File

In the `composite.xml` file, the loan application approver web service appears, as shown below.

```
<component name="LoanBroker">
    <implementation.bpel process="LoanBroker.bpel"/>
</component>
```

For more information, see Adding a Partner Link for an Asynchronous Service for instructions on creating a partner link.

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 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.

The following provides an example.

```
  <variables>
    <variable name="request"
              messageType="services:LoanServiceRequestMessage"/>
    <variable name="response"
              messageType="services:LoanServiceResultMessage"/>
  </variables>
<sequence>
    <!-- initialize the input of LoanService -->
    <assign>
    . . .
    . . .
    </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 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.

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.

The following example shows the source code for the `createInstance` attribute:

```
    <!-- receive input from requester -->
    <receive name="receiveInput" partnerLink="client"
             portType="tns:LoanBroker"
             operation="initiate" variable="input"
             createInstance="yes"/>
```

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 Dehydrate Activity.

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. The following provides an example.

```
<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.example.com:80/a.jsp
@http://myhost.us.example.com:8888/soa-infra/services/HelloWorldApp/HelloWorld!
1.0*2007-10-22_14-33-04_195/client
 </property>
</binding.ws>
</reference>
```

What You May Need to Know About Midprocess Receive Activities Consuming Messages After Timing Out

A BPEL process can consume midprocess receive activity messages even after the expiration of a configured timeout on the receive activity, if the exception resulting from the timeout goes unhandled. In these scenarios, the callback message is consumed when it is delivered. This is the expected behavior.

For example, assume you perform the following tasks:

- Create a SOA composite application with a client BPEL process and service BPEL process to exchange a message using asynchronous invoke and receive activities.
- Configure a timeout of 30 seconds in the **Timeout** tab of the receive activity of the client BPEL process.
- Configure a wait activity to wait for five minutes in the service BPEL process.

You may expect that after the timeout occurs, the client BPEL process is marked as completed in the faulted state instead of remaining in the running state, and the callback message from the service BPEL process is ignored. However, when the timeout fault is thrown on the client BPEL process, it remains in the running state. When the service BPEL process responds five minutes after the completion of the wait activity, the response is sent back to the client BPEL process and the response is consumed by the client BPEL process and reconciled with the running process instance.

What You May Need to Know About Multiple Client Components Invoking a Composite

If multiple client components invoke a SOA composite application by using its remote WSDL file, the callback response can only be retrieved by the original client calling the remote composite if it has a receive activity. When the original client does not have a receive activity and any of the subsequent clients calling the composite has a receive

activity, the response message is lost. It goes into the recovery state of the original client process.

This is the expected behavior. This is because the composite being invoked cannot tell which client has a receive activity or if the client is indeed a BPEL process service component.

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.

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.

bpelx:conversationId in BPEL 1.1

The following 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.

```
<invoke ... bpelx:conversationId="$convId2">
</invoke>
```

```
<receive ... bpelx:conversationId="$convId2">
</receive>

<onMessage... bpelx:conversationId="$convId2">
</onMessage>
```

bpelx:conversationId in BPEL 2.0

The following 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.

```
<invoke ...>
  <bpelx:conversationId>$convId1</bpelx:conversationId>
</invoke>

<receive ...>
  <bpelx:conversationId>$convId1</bpelx:conversationId>
</receive>

<onMessage ...>
  <bpelx:conversationId>$convId2</bpelx:conversationId>
</onMessage>
```

Routing Callback Messages to the Correct Endpoint when Multiple Receive or Pick Activities Use the Same Partner Link

The `replyToAddress` normalized message property is required for resolving the routing of callback messages to the correct endpoint address when multiple receive or pick activities are associated with the same partner link.

This is because the BPEL process service engine only stores the `replyToAddress` property once when receiving a request from a partner link at the initiating receive or pick activity. The `replyToAddress` property routes the callback message and is not reset if a midprocess receive or pick activity is used. The `replyToAddress` property uses the `bpelx:inputProperty` extension.

How to Route Callback Messages to the Correct Endpoint when Multiple Receive and Pick Activities Use the Same Partner Link

Set this property to the client's `replyToAddress` on the invoke activity (for the callback) following the midprocess receive activity. This means that even if the client sends WS-Addressing `replyTo` information for a midprocess receive activity, it is not set on the partner link unless you use an assign activity to set it dynamically.

For example, assume your BPEL process is as shown below:

```
Caller                                Callee
------------------------------------------------------------
<receive>                             <receive> Initiate CS1
<invoke>initiate CS1 -------->        <receive> Use CS1
                                      <wait>
<receive>use CS1    <--------         <invoke>
<invoke>
```

To route callback messages to the correct endpoint when multiple receive and pick activities use the same partner link:

1. Obtain the client's `replyToAddress` value from the midprocess receive activity.

```
<receive name="receiveMsgFromAccessor" partnerLink="midprocess_client"
       portType="client:mySingletonBPEL" operation="process"
       variable="ReceiveMidProcess" createInstance="no">
<bpelx:fromProperties>
   <bpelx:fromProperty name="replyToAddress" variable="var_replyToAddress"/>
</bpelx:fromProperties>
   <correlations>
       <correlation set="<YourCorrset>" initiate="no"/>
   </correlations>
</receive>
```

2. On the invoke activity (for the callback), click the **Properties** tab.
3. Click the **Add** icon to select the property and its content (either a variable or an XPath expression).

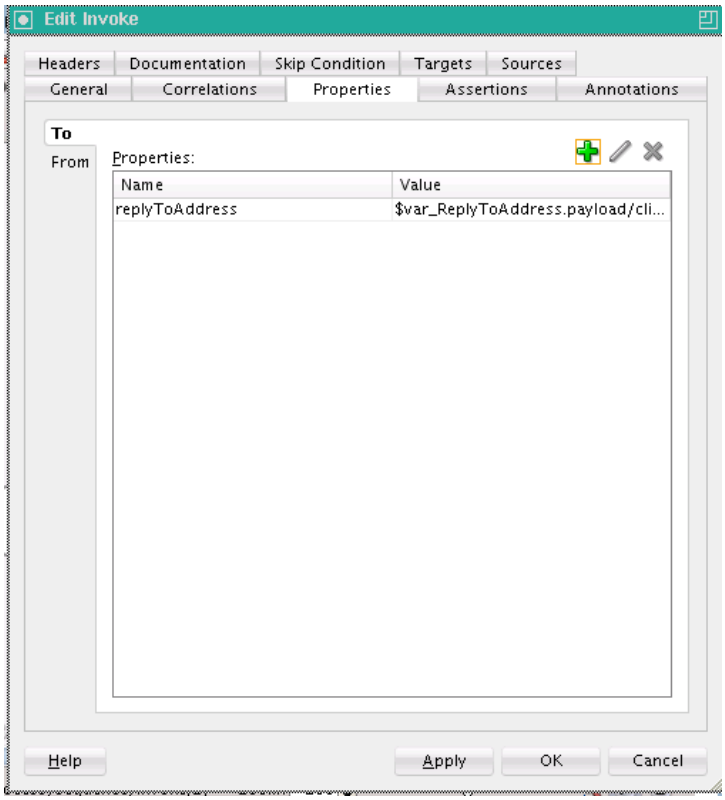
   Note:

   In BPEL 1.1 processes, the properties are automatically displayed in the **Properties** column. Select the property in the **Name** column and double-click the **Value** and **Type** columns to enter appropriate values.

4. In the **Name** column, scroll down and select the **replyToAddress** property. Do *not* select **wsa.replyToAddress** or **bpel.replyToAddress**.
5. In the **Value** column, specify the variable name as the value (for this example, `var_replyToAddress` from Step 1 is entered), and click **OK**.

   The Edit Invoke dialog appears as shown in Figure 8-1.

Figure 8-1 Properties Tab of Invoke Activity



6. Click **Apply**, then **OK**.
7. In Oracle BPEL Designer, click **Source**.

   The invoke activity in the BPEL process file looks as follows:

```
<invoke name="callbackAccessor" partnerLink="midprocess_client"
                portType="client:mySingletonBPELCallback"
                operation="processResponse"
inputVariable="CallbackAccessorVar"
                bpelx:invokeAsDetail="no">
            <bpelx:inputProperty name="replyToAddress"
variable="var_replyToAddtess"/>
```

Managing Idempotence at the Partner Link Operation Level

An idempotent activity is an activity that can be safely retried. Idempotent activities are applicable to both durable and transient processes. You can manage idempotence at the operation level of a partner link. For example, some partner links can expose multiple operations (for example, `getEmployee`, `depositPayCheck`, and so on). You can define some operations as idempotent (for example, `getEmployee`). This enables these operations to be called multiple times. Other operations may not need to be idempotent (for example, `depositPayCheck`), and do not require this setting. Dehydration does occur after a nonidempotent operation.

By default, all partner link operations are idempotent. If you want, you can set an operation to be nonidempotent. This setting provides the same functionality as the `idempotent` deployment descriptor property, but at the more granular, operational level.

For more information about the `idempotent` deployment descriptor property, see What You May Need to Know About the idempotent Property and Fault Handling and Introduction to Deployment Descriptor Properties.

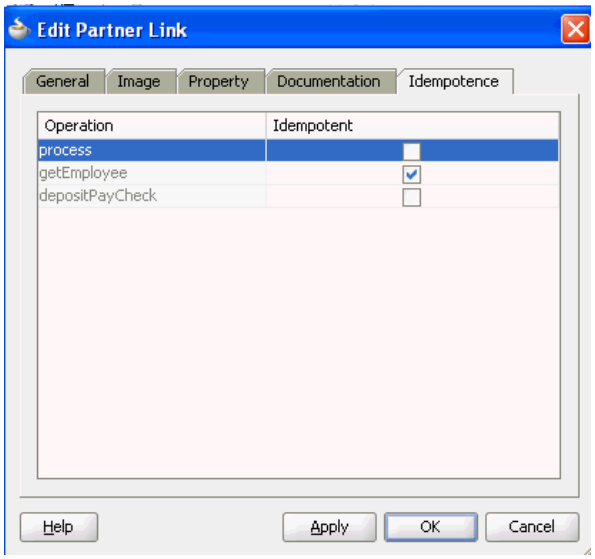
How to Manage Idempotence at the Partner Link Operation Level

To manage idempotence at the partner link operation level:

1. In Oracle BPEL Designer, double-click the partner link that includes the operations for which to manage idempotence.
2. Click the **Idempotence** tab of the partner link.

   By default, all operations are selected to be idempotent in the **Idempotent** column.
3. If you want to define an operation to be nonidempotent, deselect the **Idempotent** check box for that operation. Figure 8-2 provides details.

Figure 8-2 Idempotence Tab of Partner Link Activity



4. Click **Apply**.
5. Click **OK**.

For more information about idempotence and the `idempotent` property, see Introduction to Deployment Descriptor Properties.

Creating a Dynamic Partner Link at Design Time for Use at Runtime

When you design a SOA composite application, you can face the following challenges:

- Service endpoints (addresses) may not be known at design time.
- Endpoint references may need to change while the application is running.

The dynamic partner link feature enables you to dynamically assign an endpoint reference to a partner link for use at runtime in BPEL versions 1.1 and 2.0. The dynamic partner link provides conditions, similar to a switch activity, that are evaluated at runtime.

How To Create a Dynamic Partner Link at Design Time for Use at Runtime

To create a dynamic partner link at design time for use at runtime:

1. Create a WSDL file that contains multiple services that use the same `portType`.

```
<service name="AmericanLoan">
 <port name="LoanServicePort" binding="tns:LoanServiceBinding">
  <soap:address location="host:port/soa-infra/services/domain_
name/AmericanLoan/client"/>
 </port>
</service>

<service name="AlliedLoan">
 <port name="LoanServicePort" binding="tns:LoanServiceBinding">
  <soap:address location="host:port/soa-infra/services/domain_
name/AlliedLoan/client"/>
 </port>
</service>

<service name="AcmeLoan">
 <port name="LoanServicePort" binding="tns:LoanServiceBinding">
  <soap:address location="host:port/soa-infra/services/domain_
name/AcmeLoan/client"/>
 </port>
</service>
```

2. Drag a **SOAP** binding component into the **External References** swim lane of the SOA Composite Editor.

   The Create Web Service dialog appears.

3. Define the web service, and click **OK**.

   When complete, the reference binding component entry in the `composite.xml` file that uses the WSDL looks as follows:

```
<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>
```

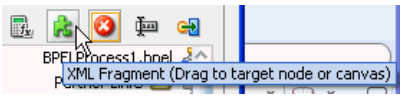
   Note:

   - Adding the `binding.ws port` setting is optional. This is because the port is overridden at runtime by properties passed from Oracle BPEL Process Manager.
   - 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.

4. Double-click the BPEL process to enter Oracle BPEL Designer.

5. Drag an **Assign** activity into the designer.
6. Above the target partner link, select the **XML Fragment** icon, as shown in Figure 8-3. If you are using BPEL 2.0, drag the **Literal** icon.

Figure 8-3 XML Fragment Icon

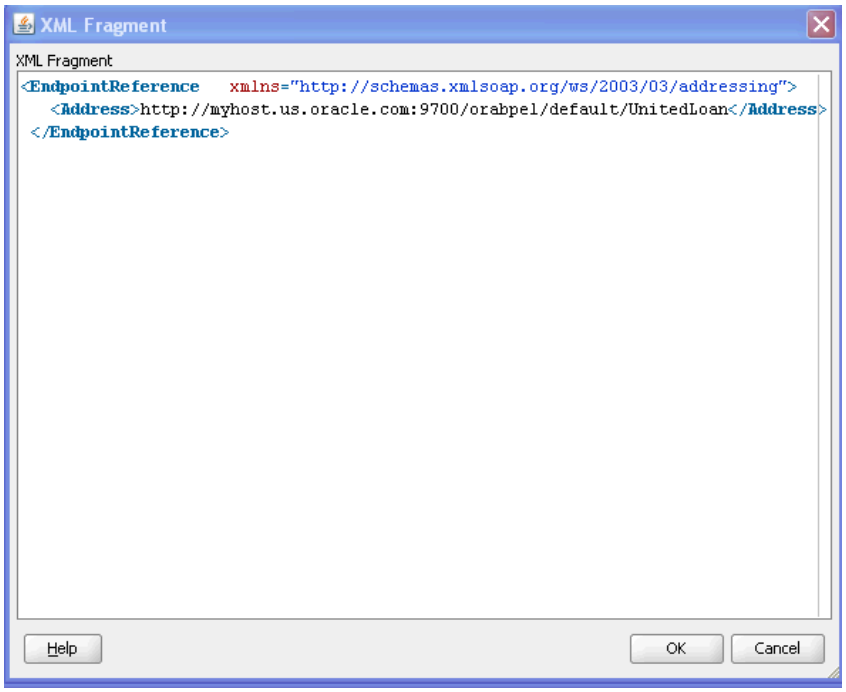


7. Drag the icon to the target partner link.

   The XML Fragment dialog for BPEL 1.1 appears. If you are using BPEL 2.0, the Literal dialog appears.

8. Assign an XML fragment containing the endpoint reference to the partner link, and click **OK**. Figure 8-4 provides details.

Figure 8-4 XML Fragment Dialog in BPEL 1.1



When complete, the BPEL file contains one of the services defined in the WSDL.

The following provides a BPEL 1.1 sample:

```
<EndpointReference xmlns="http://schemas.xmlsoap.org/ws/2003/03/addressing">
         <Address>http://host:port/soa-infra/services/domain_name
           /AlliedLoan/client</Address>
<ServiceName xmlns:ns1="http://services.otn.com"
   PortName="LoanServicePort">ns1:AlliedLoan</ServiceName>
</EndpointReference>
```

The following provides a BPEL 2.0 sample:

```
<assign>
  <copy>
    <from>
      <literal>
```

```
        <sref:service-ref>
          <services:EndpointReference>
          <services:Address>http://host:port/soa-infra/services/domain_
           name/AlliedLoan/client</services:Address>
           <services:ServiceName
           xmlns:ns1="http://services.otn.com">ns1:AlliedLoan</services:
           ServiceName>
          </services:EndpointReference>
        </sref:service-ref>
      </literal>
    </from>
    <to partnerLink="LoanService"/>
  </copy>
</assign>
```

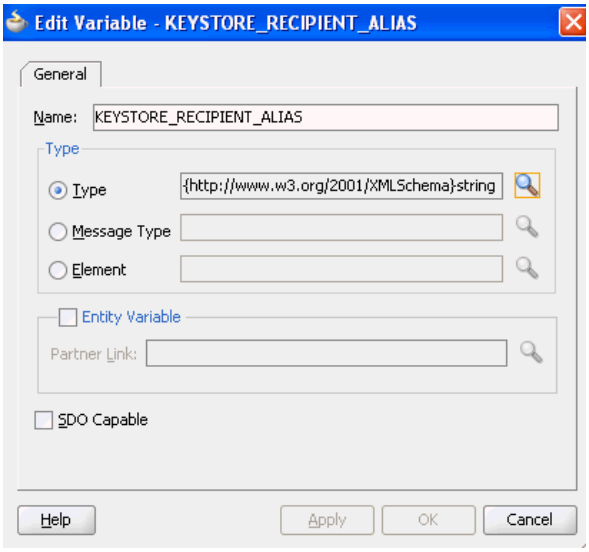
Overriding Security Certificates when Invoking Dynamic Partner Links

You can interact with multiple web services using dynamic partner links. This interaction may involve using message protection policies that require different security certificates for encrypting the message. These certificates may be different for each web service. You can specify a keystore recipient alias value to override the security certificate in the WSDL file of the web service.

To override security certificates when invoking partner links:

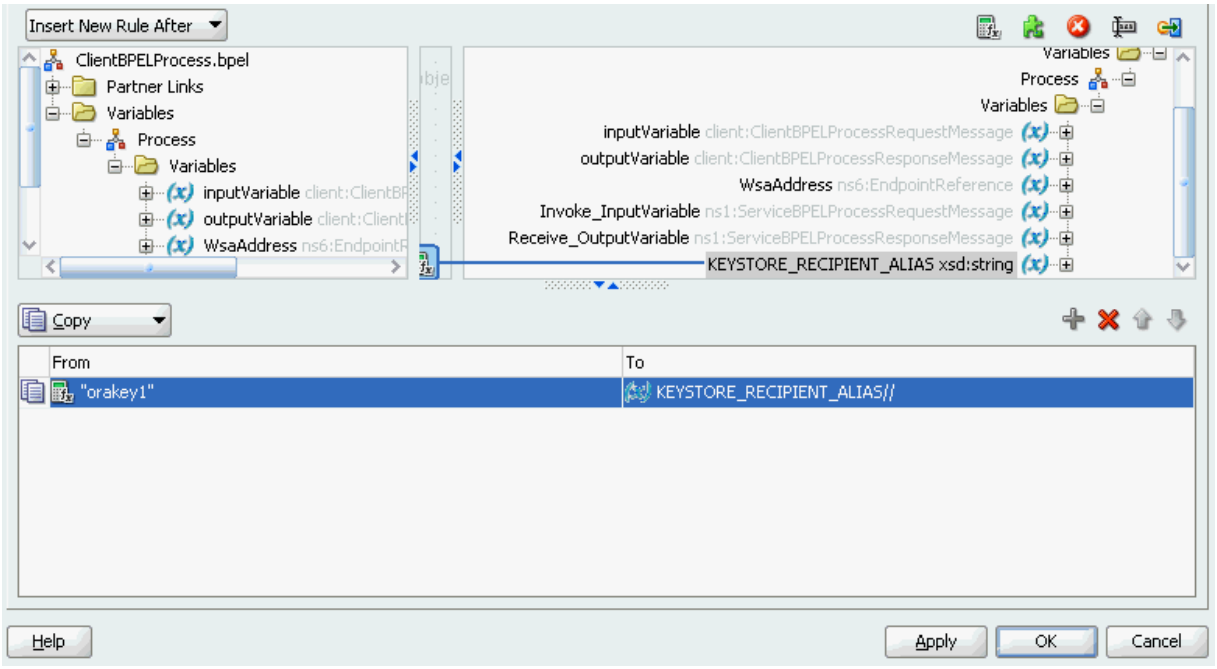
1. Define a variable of type string (for example, **KEYSTORE_RECIPIENT_ALIAS**). Figure 8-5 provides details.

Figure 8-5 Variable Definition of KEYSTORE_RECIPIENT_ALIAS



2. In the **Copy Rules** tab of an assign activity, assign **orakey** to the variable **KEYSTORE_RECIPIENT_ALIAS**. Figure 8-6 provides details.

Figure 8-6 Assignment of orakey to KEYSTORE_RECIPIENT_ALIAS



3. In the invoke activity that invokes the partner link for the web service, click the **Properties** tab.
4. Click the **keystore.recipient.alias** property.

Note:

In BPEL 2.0 processes, the properties are not automatically displayed in the **Properties** column. You must click the **Add** icon to select the property and its content (either a variable or an XPath expression).

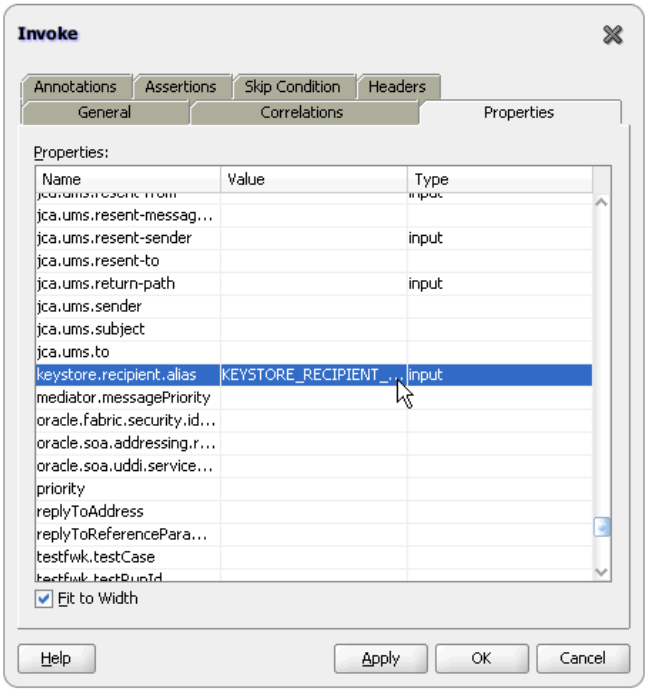
---

5. Double-click the **Value** column to display the **Browse** (...) icon.
6. Click the **Browse** (...) icon to display the Adapter Property Value dialog.
7. Click the **Browse** icon to display the Variable XPath Builder dialog.
8. Select **keystore_recipient_alias** as the value, and click **OK**. Figure 8-7 provides details. This property overrides the security certificates set in the WSDL file while invoking a web service in a BPEL process.

Note:

In BPEL 2.0, there are only **Name** and **Value** columns in the **Properties** table. The **Type** column is not included.

Figure 8-7 keystore.recipient.alias Normalized Message Property of Invoke Activity



9. Click **Apply**, then **OK**.

When complete, the BPEL file is defined as follows:

```
. . .
. . .
<variables>
  <variable name="WsaAddress" element="ns6:EndpointReference"/>
  <variable name="KEYSTORE_RECIPIENT_ALIAS" type="xsd:string"/>
</variables>

<assign name="AssignAddress">
  <copy>
    <from
expression="'http://localhost:8001/soa-infra/services/default/ServiceWithNewCer
tificate!1.0*soa_c94537fb-97a4-4b0f-900f-fefffc34f7fe/service_ep'"/>
    <to variable="WsaAddress"
    query="/ns6:EndpointReference/ns6:Address"/>
  </copy>
  <copy>
    <from variable="WsaAddress"/>
    <to partnerLink="Service"/>
  </copy>
</assign>

<assign name="AssignAlias">
  <copy>
    <from expression='"orakey"'/>
    <to variable="KEYSTORE_RECIPIENT_ALIAS"/>
  </copy>
</assign>

<invoke name="Invoke"
        inputVariable="Invoke_InputVariable"
```

```
            partnerLink="Service"
            portType="ns1:ServiceBPELProcess"
            operation="process"
            bpelx:invokeAsDetail="no">

                <bpelx:inputProperty name="endpointURI"
                                     variable="inputVariable"
                                     part="payload"
                                     query="/client:process/client:input"/>

                <bpelx:inputProperty name="keystore.recipient.alias"
                                     variable="KEYSTORE_RECIPIENT_ALIAS"/>
    </invoke>
```

For more information about normalized message properties, see Propagating Normalized Message Properties Through Message Headers.

Overriding WSDL Files of Dynamic Partner Links

You may need to override the default WSDL file used by dynamic partner links for the following reasons:

- You must integrate with services that use message protection security policies.
- The WSDL may contain important information such as the certificate used for message encryption.

The normalized message property **endpointWSDL** enables you to specify the WSDL file of the dynamic partner link. You must specify the entire WSDL dynamically instead of just the endpoint. This enables it to be passed to Oracle Web Services Manager (OWSM), which can then retrieve the correct service certificate from the specified WSDL.

The certificate in the WSDL file is ignored in the following cases:

- The **recipient.key.alias** property name described in Overriding Security Certificates when Invoking Dynamic Partner Links is present.
- The **endpointWSDL** property is not present.

Otherwise, the certificate is retrieved from the WSDL file.

To override WSDL files of dynamic partner links:

1. Define a variable of type string (for this example, `the_wsdl_var` is defined).
2. In the **Copy Rules** tab of an assign activity, assign the WSDL to **the_wsdl_var**.
3. In the invoke activity that invokes the partner link, click the **Properties** tab.
4. Click the **endpointWSDL** property.

Note:

In BPEL 2.0 processes, the properties are not automatically displayed in the **Properties** column. You must click the **Add** icon to select the property and its content (either a variable or an XPath expression).

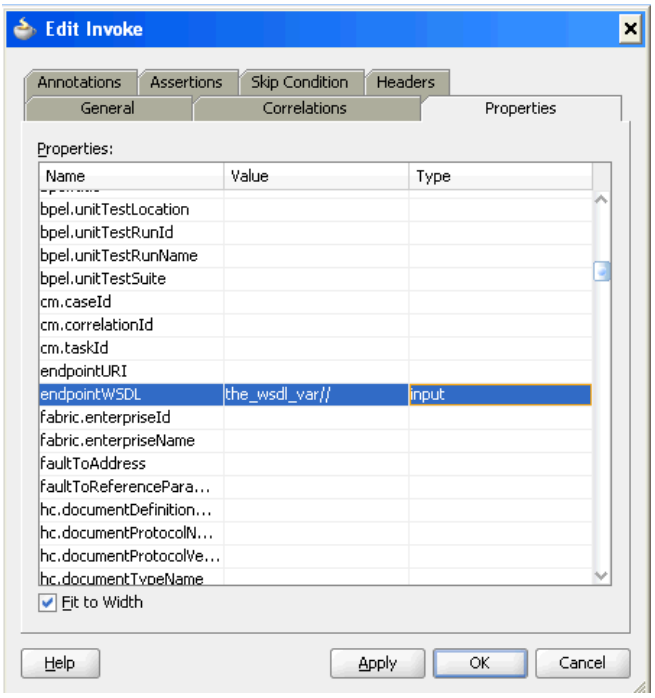
---

5. Double-click the **Value** column to display the **Browse** (...) icon.
6. Click the **Browse** (...) icon to display the Adapter Property Value dialog.
7. Click the **Browse** icon to display the Variable XPath Builder dialog.
8. Select **the_wsdl_var** as the variable, and click **OK**. This value specifies the WSDL of the dynamic partner link.

Note:

In BPEL 2.0, there are only **Name** and **Value** columns in the **Properties** table. The **Type** column is not included.

Figure 8-8 endpointWSDL Normalized Message Property of Invoke Activity



When complete, the BPEL file is defined as follows:

```
<variables>
  <variable name="the_wsdl_var" type="xsd:string"/>
</variables>

<assign name="myAssignWsdl">
  <copy>
    <from
expression='"http://localhost:8001/soa-infra/services/default/ServiceWithNewCer
tificate!1.0/service_ep?WSDL"'/>
    <to variable="the_wsdl_var"/>
  </copy>
</assign>

<invoke name="Invoke"
        inputVariable="Invoke_InputVariable"
        partnerLink="Service"
        portType="ns1:ServiceBPELProcess"
```

```
            operation="process"
            bpelx:invokeAsDetail="no">

                <bpelx:inputProperty name="endpointWSDL"
                                     variable="the_wsdl_var"/>

    </invoke>
```

For more information about normalized message properties, see Propagating Normalized Message Properties Through Message Headers.

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 8-9 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 8-9 Callback with WS-Addressing Headers

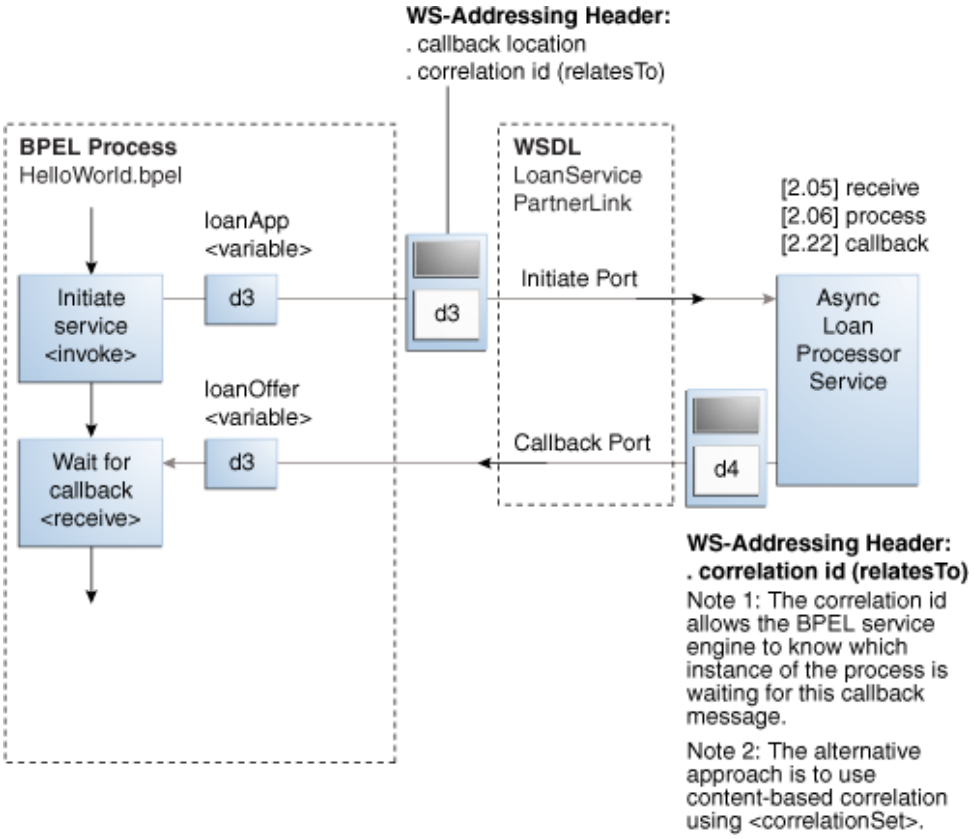


Figure 8-9 shows how messages are passed along with WS headers so that the response can be sent to the correct destination.

The example in this chapter uses WS-Addressing for correlation. To view the messages, you can use TCP tunneling, which is described in Using TCP Tunneling to View 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.

How to Use WS-Addressing in an Asynchronous Service

WS-Addressing is a public specification and is the default correlation method supported by . You do not need to edit the `.bpel` and `.wsdl` files to use WS-Addressing.

Using TCP Tunneling to View 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 view 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.

Setting Up a TCP Listener for Synchronous Services

Follow these steps to set up a TCP listener for synchronous services initiated by an 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`.

   The same technique can see SOAP messages passed to invoke a BPEL process service component as a web service from another tool kit such as Axis or .NET.

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 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 navigate to the specific **SCAComposite.SCAReference.SCABinding** folder.

   b. Click **WSBinding**.

   c. Perform steps 44.b through 44.l.

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.

For information about using correlation sets for message correlation, see Using Correlation Sets and Message Aggregation .

9

Using Correlation Sets and Message Aggregation

This chapter describes how to use correlation sets to ensure that asynchronous callbacks locate the appropriate client. It also describes how to use aggregation patterns to route messages to the same instance.

This chapter includes the following sections:

Introduction to Correlation Sets in an Asynchronous Service

Correlation sets provide a 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. You define correlation sets when interactions are not simple invoke-receive activities.

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 process. 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`.

Scenarios for Using Correlation Sets

Correlations enable you to associate asynchronous messages based on message body contents. Note that not all business scenarios require correlations:

- Synchronous calls do not require correlations because the conversation context is maintained in the stack or across a TCP connection.
- Consenting BPEL processes typically correlate messages using WS-Addressing headers to pass tokens that act like session cookies in a web application. For more information, see Using WS-Addressing in an Asynchronous Service.

Correlation is required in the following scenarios. In these cases, a BPEL process must be configured to view some content of the message to select the correct process instance to receive the message.

- When using an asynchronous service that does not support WS-Addressing.
- When receiving unsolicited messages from another system.

- When the message travels through several services and the response is solicited by the initial service from the last service directly.
- When communicating through files.

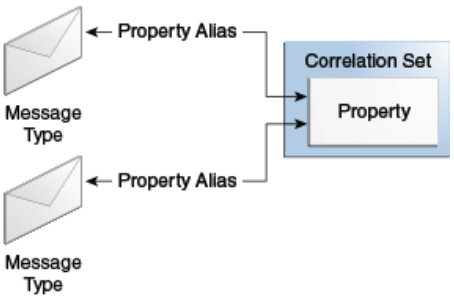
Understanding Correlation Set Contents and Concepts

This section provides an overview of key correlation set concepts.

The correct BPEL instance using correlation sets is obtained as follows:

- A BPEL process provides a construct called a correlation set to allow for custom correlation.
- A correlation set is a collection of properties used by the BPEL process service engine to identify the correct process to receive a message.
- Each property in the correlation set can be mapped to an element in one or more message types through property aliases. Figure 9-1 provides an overview.

Figure 9-1 Correlation Sets



Note the following key correlation guidelines:

- Only the process receiving the message is concerned about correlation. As long as the sending service includes sufficient information in the message to correlate it with previous activities, the sender does not need to be aware that correlation is occurring.
- Correlation properties must be unique for the duration of the life of the BPEL process that sets them.
- Ensure that no two processes are working with the same correlation tokens. For example, using social security numbers to correlate an expense claims process is not recommended if you start two separate instances of the process.
- Properties can be made up values or actual business identifiers such as purchase orders or numbers. They do not need to be strings; they can be any reasonable XML type.

Key correlation concept attributes are as follows. You set these attributes in Oracle JDeveloper when designing a correlation set with the Correlation wizard:

- An **initiate** attribute is set as follows:
  - **yes**: The correlation set is initiated with the values of the properties available in the message being transferred.

  - **no**: The correlation set validates the value of the property available in the message.

- A **pattern** attribute is set as follows:
  - **in** (for BPEL 1.1) or **response** (for BPEL 2.0): The correlation property is set and validated on the incoming message.
  - **out** (for BPEL 1.1) or **request** (for BPEL 2.0): The correlation property is set and validated on the outgoing BPEL message.
  - **out-in** (for BPEL 1.1) or **request-response** (for BPEL 2.0): The correlation property is set and validated on both incoming and outgoing messages.
- Property aliases 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.

Overview of Correlation Set Creation

Table 9-1 provides an overview of the steps for creating a correlation set. References to the pages of the Correlation wizard on which you perform these steps and examples of values to set are provided.

Table 9-1 Correlation Set Creation Overview

Step	Correlation Wizard Page	Example
Create a correlation set with property names and types to correlate the exchange.	Set this information on the Correlation wizard - Define Correlation Set page. See Figure 9-2.	Create a **phonenumber** correlation set with property names and types: • **username** of type **string** • **userordernumber** of type **int** • **IsGift** of type **boolean**
Add the correlation to the invoke or receive activity that begins the conversation and set **Initiate** to **yes**.	Select the activity and set the **Initiate** attribute on the Correlation wizard - Initiate Settings page. See Figure 9-3.	Select the **internalReceive** receive activity and set **Initiate** to **yes**.
Create property alias mappings to appropriate elements in each message. They must have the same value in both messages of the conversation. The elements can be different names and in different structures in the two messages, but they must contain the same value for correlation to work.	Set this information on the Correlation wizard - Property Aliases page. See Figure 9-7. Two editors available on this page enable you to create the property alias mappings: • Alias Editor (Figure 9-4) • Alias Drag and Drop Editor (Figure 9-5)	Define the property aliases to populate the correlation set property values at runtime: • Map alias **username** to the **name** message element • Map alias **userordernumber** to the **poNumber** message element • Map alias **IsGift** to the **gift** message element.

Table 9-1 (Cont.) Correlation Set Creation Overview

Step	Correlation Wizard Page	Example
Add the same correlation set with its property to additional activities. Do not set them to **initiate**. The BPEL process uses this to select the correct process instance. Set the pattern accordingly.	Set on the Activity Correlation Editor - Initiate Tab. See Figure 9-10.	Select the **internalCallback** invoke activity: • Set **Initiate** to **no** • Set **Pattern** to **request**

Creating Correlation Sets in Oracle JDeveloper

You can create correlation sets on the following activities and branches.

- Receive activity
- Reply activity
- Invoke activity
- onMessage branch
- onEvent branch

There are two methods for creating correlations sets in Oracle JDeveloper:

- Automatically through the Correlation wizard in an activity
- Manually through the **Correlations** tab in an activity

How to Create a Correlation Set with the Correlation Wizard

To create a correlation set with the Correlation wizard:

1. Right-click an applicable activity (such as a receive activity), and select **Setup Correlation**.

   The Correlation wizard - Define Correlation Set page is displayed.

2. Provide responses appropriate to your environment, then click **Next**. Table 9-2 provides details.

Table 9-2 Correlation Wizard - Define Correlation Set Page

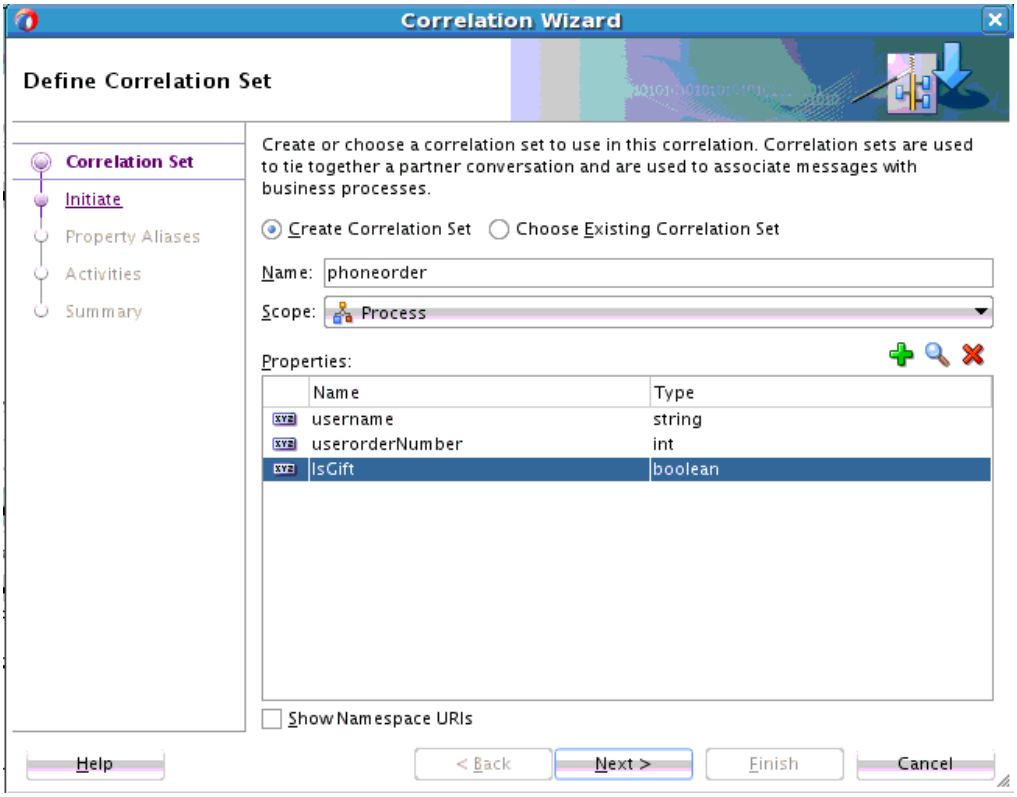
Field	Description
Create Correlation Set	Select to create a new correlation set.
Choose Existing Correlation Set	Select an existing correlation set in which to include the selected activity.
Name	Enter the name of the correlation set you want to create.

Table 9-2 (Cont.) Correlation Wizard - Define Correlation Set Page

Field	Description
Scope	Displays the scope or process in which to create the new correlation set.
Properties	**a.** Click **Add** to create a new property in the **Name** column of the **Properties** table or click **Browse** to select an existing property. **b.** Click the **Type** column, then click the ellipses to invoke the Type Chooser dialog for selecting the property type (for example, integer, boolean, or some other type).

When complete, the Correlation wizard - Define Correlation Set page looks as shown in Figure 9-2.

Figure 9-2 Correlation Wizard - Define Correlation Set Page



The Correlation wizard - Initiate Settings page is displayed.

3. Provide responses appropriate to your environment, then click **Next**. Table 9-3 provides details.

Table 9-3 Correlation Wizard - Initiate Settings Page

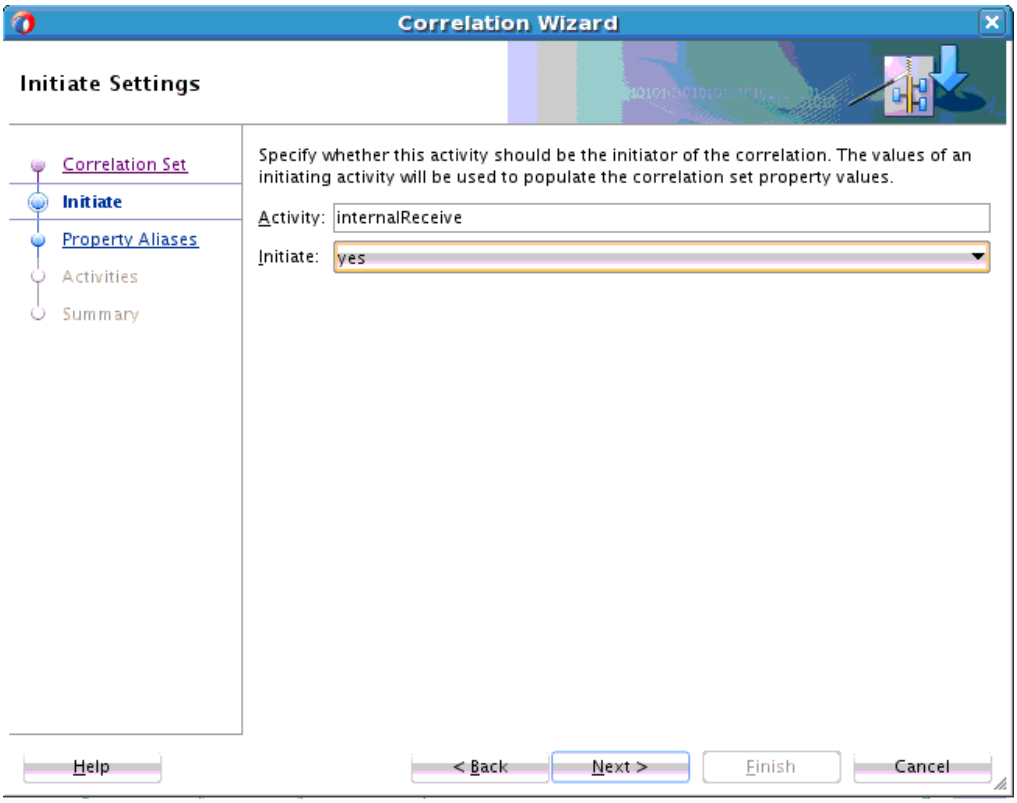
Field	Description
Activity	Displays the activity on which the correlation is set.

Table 9-3 (Cont.) Correlation Wizard - Initiate Settings Page

Field	Description
initiate	Select whether this activity is the initiator in the correlation set. When set to **yes**, the correlation set is initiated with the values of the properties occurring in the message being sent or received.

When complete, the Correlation wizard - Initiate Settings page looks as shown in Figure 9-3.

Figure 9-3 Correlation Wizard - Initiate Settings Page



The Correlation wizard - Property Aliases page is displayed for mapping properties to values. The properties defined previously in the Define Correlation Set page of the wizard are displayed in the **Property Aliases** table.

Property aliases enable you to map a property to a field in a specific message part of a variable. This action enables the property to become an alias for the message part and location.

4. Click a property in the table and select a method for mapping the message part of the variable to the property. Table 9-4 provides details.

Table 9-4 Methods for Mapping the Variable Message Part to a Property

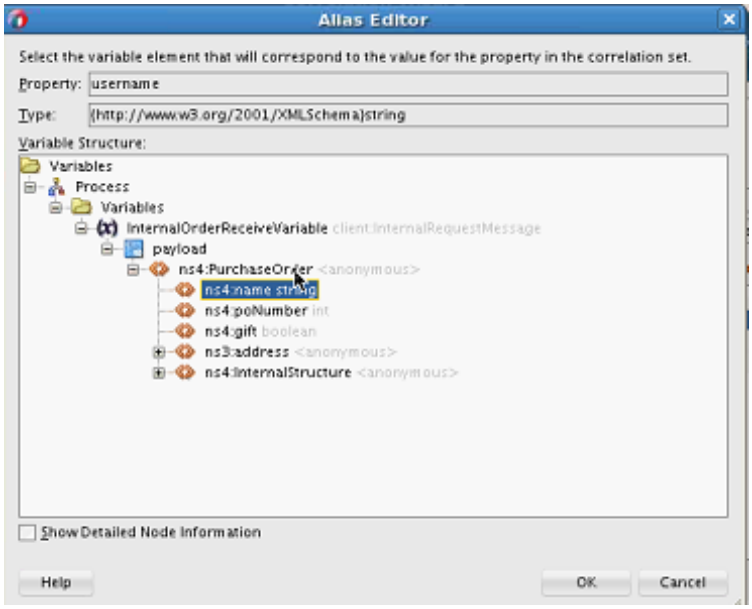
To Use The...	Go to Step...
Alias Editor	5
Alias Drag and Drop Editor	6

5. Click the **Edit** (first) icon to invoke the Alias Editor dialog.

   a. Expand the variable.

   b. Select the message part to represent the property, and click **OK**. Figure 9-4 provides details.

Figure 9-4 Alias Editor



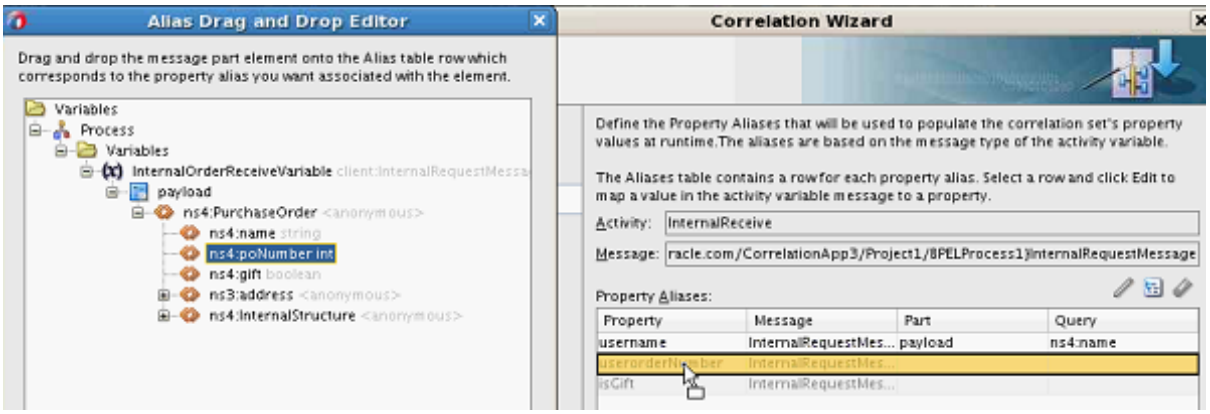
6. Click the **Alias Drag and Drop Editor** (second) icon to invoke the Alias Drag and Drop Editor dialog.

   a. Expand the variable.

   b. Select the message part to represent the property.

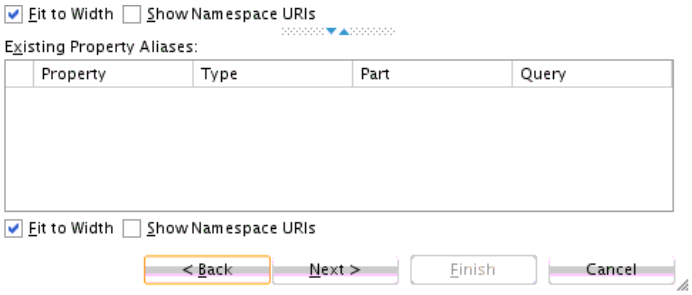
   c. Drag and drop the message part onto the property row in the Correlation wizard - Property Aliases page. Figure 9-5 provides details.

Figure 9-5 Alias Drag and Drop Editor



Existing property aliases are listed in the lower part of the Correlation wizard - Property Aliases page, as shown in Figure 9-6. For this example, there are no existing property aliases.

Figure 9-6 Correlation Wizard - Property Aliases Page - Lower Part



d. When complete, click **Next**.

7. Select additional properties to map to specific message parts of variables.

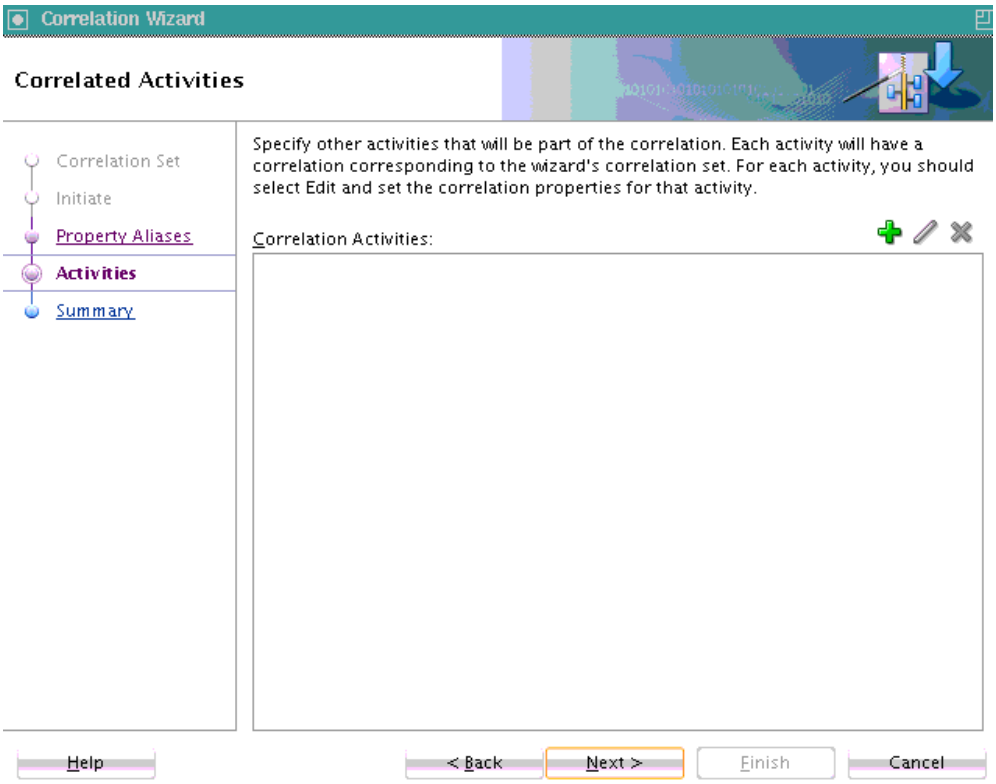
When complete, the Correlation wizard - Property Aliases page looks as shown in Figure 9-7. The properties created in Figure 9-2 are displayed in the **Property** column. The message elements to which the properties were mapped with either the Alias Editor (Figure 9-4) or Alias Drag and Drop Editor (Figure 9-5) are displayed in the **Query** column.

Figure 9-7 Correlation Wizard - Property Aliases Page

8. Click **Next**.

The Correlation wizard - Correlated Activities page is displayed. Figure 9-8 provides details.

Figure 9-8 Correlation Wizard - Property Aliases Page (Without Activity)



9. Click the **Add** icon to add more activities to this correlation set (multiple activities can correlate on the correlation set).

   The Activity Browser dialog is displayed.

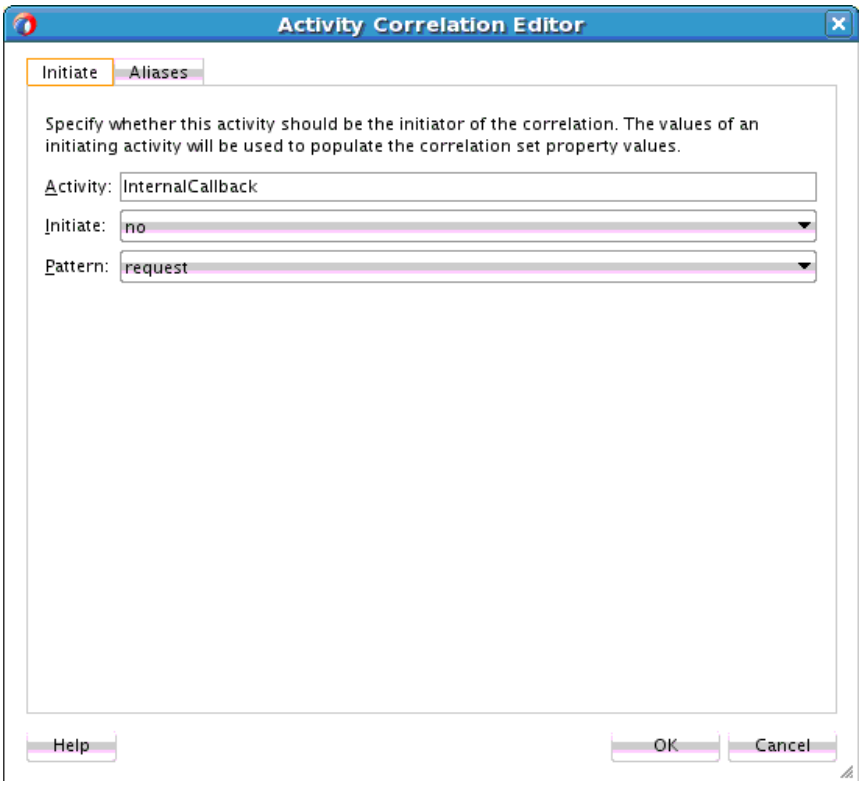
10. Select the activity to add, and click **OK**. Figure 9-9 provides details.

Figure 9-9 Activity Browser for Selecting an Activity

The activity is added to the **Correlation Activities** field of the Correlation wizard - Correlated Activities page.

11. In the **Correlation Activities** field, select the activity and click **Edit** to invoke the **Initiate** tab of the Activity Correlation Editor dialog. Figure 9-10 provides details.

Figure 9-10 Activity Correlation Editor - Initiate Tab



12. Select appropriate values in the **Initiate** and **Pattern** lists. For this example:

    - Select **no** from the **Initiate** list (because the correlation set validates the value of the property available in the message).
    - Select **request** from the **Pattern** list (because the correlation property is set and validated on the outgoing BPEL message).

    For BPEL 2.0, you can select **response** if the correlation applies to an inbound message, **request** if the correlation applies to an outbound message, or **request-response** if the correlation applies to both outbound and inbound messages.

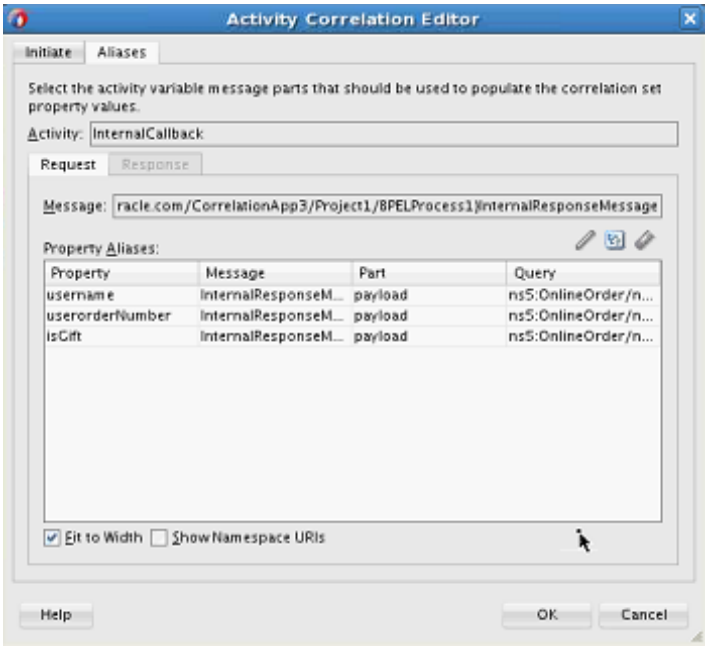
    For BPEL 1.1, you can select **in** if the correlation applies to an inbound message (response), **out** if the correlation applies to an outbound message (request), or **out-in** if the correlation applies to both inbound and outbound messages. (response and request).

13. Click the **Aliases** tab.

14. Repeat Step 4 through Step 7 to select a property and map the message part of the variable to the property.

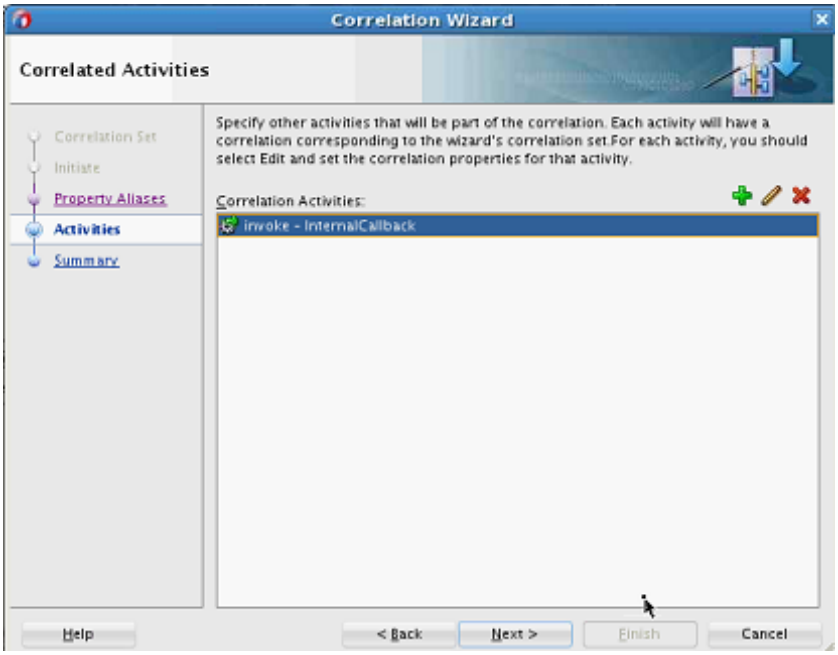
    When complete, the Alias dialog looks similar to that shown in Figure 9-11.

Figure 9-11 Activity Correlation Editor - Alias Tab



15. Click **OK** to return to the Correlation wizard - Correlated Activities page, which looks as shown in Figure 9-12.

Figure 9-12 Correlation Wizard - Correlated Activities Page (With Selected Activity)



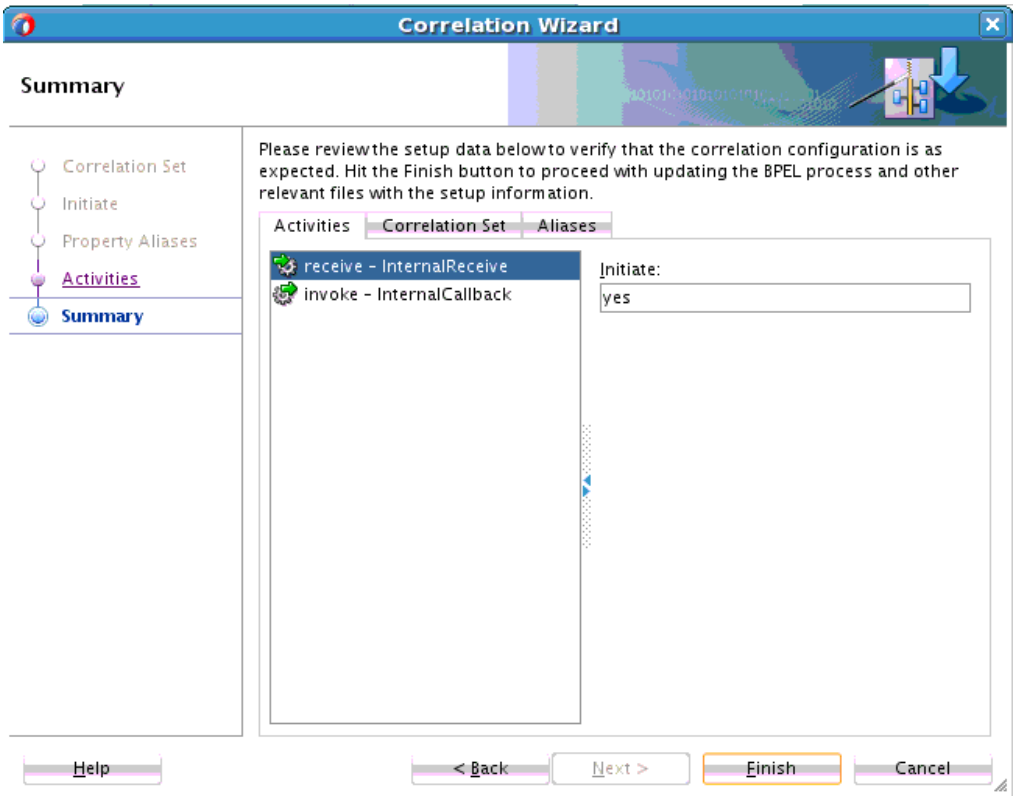
16. Click **Next** to review the correlation set details in the **Activities**, **Correlation Set**, and **Alias** tabs.

    - **Activities**: Displays the activities involved in the correlation and their roles (for example, the receive activity is the initiator and the invoke activity is the responder).

- **Correlation Set**: Displays the name of the correlation set.
- **Aliases**: Displays the property aliases defined for the activities in the correlation set.

Figure 9-13 provides details.

Figure 9-13 Correlation Wizard - Summary Page



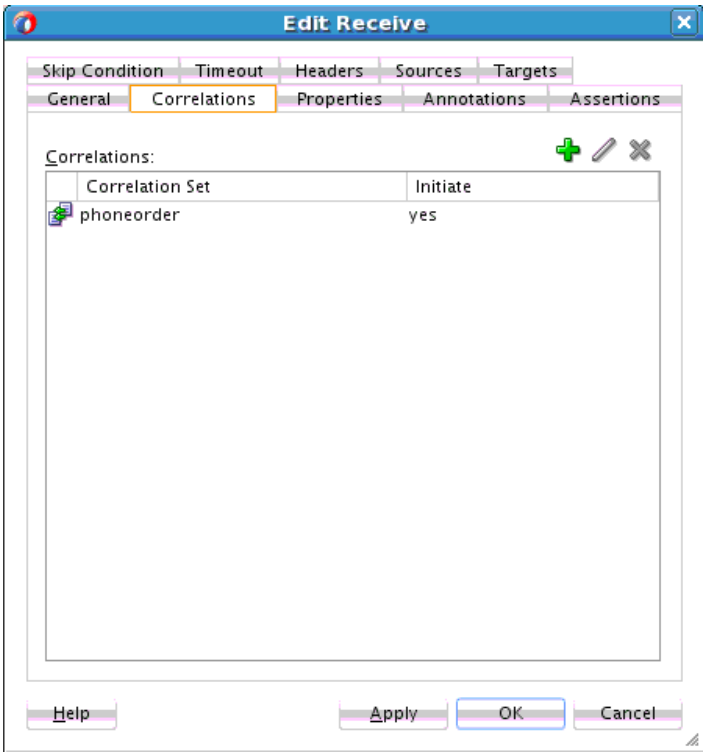
17. Click **Finish**.

    The correlation set is created.

18. In the Structure window, view the correlation set, properties, and property aliases you defined in the Correlation wizard.

19. In Oracle BPEL Designer, click the **Correlations** tab of one of the participating activities to view the details you defined (for example, the receive activity). Figure 9-14 provides details.

Figure 9-14 Correlation Tab of Receive Activity



20. If you want to find out which activities are used in a correlation set, perform the following steps.

    a. Click the **Search** icon above Oracle BPEL Designer, and select **Correlation Search**.

       The Correlation Set Chooser dialog is displayed.

    b. Select the correlation set, and click **OK**.

    c. In the Correlation Search dialog, click **OK**.

       The activities using the correlation sets are displayed in the Log window.

21. If you want to add additional activities to an existing correlation set, right-click the activity, and select **Setup Correlation**.

    The Correlation wizard - Define Correlation Set page is displayed.

22. Select **Choose Existing Correlation Set**.

23. From the **Correlation Sets** list, select the correlation set, and click **OK**.

24. Define the activity by following the pages in the Correlation wizard.

How to Manually Create Correlation Sets From the Correlations Tab

This section describes the steps to manually create correlation sets in an asynchronous service. This example illustrates how to use correlation sets for a process having three receive activities with no associated invoke activities.

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 a name (for this example, `MyCorrelationSetApp` is entered).
5. Accept the default values for all remaining settings, and click **Next**.
6. In the **Project Name** field, enter a name (for this example, `MyCorrelationSetComposite` is entered).
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 9-5.

Table 9-5 Create BPEL Process Dialog Fields and Values

Field	Value
Name	Enter a name (for this example, `MyCorrelationSet` is entered).
Template	Select **Asynchronous BPEL Process**.
Expose as a SOAP Service	Select the check box. 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**.

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.

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 Components window, 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 9-15.

   Figure 9-15 Adapter Configuration Wizard Startup

5. In the Configure Service or Adapter dialog, select **File** and click **OK**.
6. In the **Name** field of the File Adapter Reference dialog, enter a name (for this example, `FirstReceive` is entered) and click **Next**.
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** and click **Next**. The **Operation Name** field is automatically filled in with **Read**.
9. Above the **Directory for Incoming Files (physical path)** field, click **Browse**.
10. Select a directory from which to read files (for this example, **C:\files \receiveprocess\FirstInputDir** is selected).
11. Click **Select**.
12. Click **Next**.
13. In the File Filtering dialog, enter appropriate file filtering parameters.
14. Click **Next**.
15. In the File Polling dialog, enter appropriate file polling parameters.
16. Click **Next**.
17. In the Messages dialog, click **Browse** next to the **URL** field to display the Type Chooser dialog.
18. Select an appropriate XSD schema file. For this example, **Book1_4.xsd** is the schema and **LoanAppl** is the schema element selected.
19. Click **OK**.

    The **URL** field (**Book1_4.xsd** for this example) and the **Schema Element** field (**LoanAppl** for this example) are filled in.
20. Click **Next**.

21. Click **Finish**.

    You are returned to the Partner Link dialog. All other fields are automatically completed. The dialog looks as shown in Table 9-6:

Table 9-6 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**

22. Click **OK**.

Creating a Second Partner Link and File Adapter Service

To create a second partner link and file adapter service:

1. Drag a second **Partner Link** 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** and click **OK**.
4. In the **Name** field of the File Adapter Reference dialog, enter a name (for this example, `SecondFileRead` is entered) and click **Next**. This name must be unique from the one you entered in Step 6 of Creating an Initial Partner Link and File Adapter Service.
5. In the Adapter Interface dialog, accept the default settings and click **Next**.
6. In the Operation dialog, select **Read File** as the **Operation Type**.
7. In the **Operation Name** field, change the name (for this example, `Read1` is entered).
8. Click **Next**.
9. Select **Directory Names are Specified as Physical Path**.
10. Above the **Directory for Incoming Files (physical path)** field, click **Browse**.
11. Select a directory from which to read files (for this example, **C:\files \receiveprocess\SecondInputDir** is entered).
12. Click **Select**.
13. Click **Next**.
14. Enter appropriate file filtering parameters in the File Filtering dialog.
15. Click **Next**.

16. Enter appropriate file polling parameters in the File Polling dialog.
17. Click **Next**.
18. Next to the **URL** field in the Messages dialog, click **Browse** to display the Type Chooser dialog.
19. Select an appropriate XSD schema file. For this example, **Book1_5.xsd** is the schema and **LoanAppResponse** is the schema element selected.
20. Click **OK**.

    The **URL** field (**Book1_5.xsd** for this example) and the **Schema Element** field (**LoanAppResponse** for this example) are filled in.
21. Click **Next**.
22. Click **Finish**.

    You are returned to the Partner Link dialog. All other fields are automatically completed. The dialog looks as shown in Table 9-7:

Table 9-7 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**

23. Click **OK**.

Creating a Third Partner Link and File Adapter Service

To create a third partner link and file adapter service:

1. Drag a third **Partner Link** 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** and click **OK**.
4. In the **Name** field of the File Adapter Reference dialog, enter a name (for this example, `ThirdFileRead` is entered) and click **Next**. This name must be unique from the one you entered in Step 6 of Creating an Initial Partner Link and File Adapter Service and Step 4 of Creating a Second Partner Link and File Adapter Service.
5. In the Adapter Interface dialog, accept the default settings and click **Next**.
6. In the Operation dialog, select **Read File** as the **Operation Type**.

7. In the **Operation Name** field, change the name (for this example, `Read2` is entered). This name must be unique.
8. Click **Next**.
9. Select **Directory Names are Specified as Physical Path**.
10. Above the **Directory for Incoming Files (physical path)** field, click **Browse**.
11. Select a directory from which to read files (for this example, **C:\files \receiveprocess\ThirdInputDir** is entered).
12. Click **Select**.
13. Click **Next**.
14. Enter appropriate file filtering parameters in the File Filtering dialog.
15. Click **Next**.
16. Enter appropriate file polling parameters in the File Polling dialog.
17. Click **Next**.
18. Next to the **URL** field in the Messages dialog, click **Browse** to display the Type Chooser dialog.
19. Select an appropriate XSD schema file. For this example, **Book1_6.xsd** is the schema and **CustResponse** is the schema element selected.
20. Click **OK**.

    The **URL** field (**Book1_6.xsd** for this example) and the **Schema Element** field (**CustResponse** for this example) are filled in.
21. Click **Next**.
22. Click **Finish**.

    You are returned to the Partner Link dialog. All other fields are automatically completed. The dialog looks as shown in Table 9-8:

Table 9-8 Partner Link Dialog Fields and Values

Field	Value
Name	**ThirdReceive**
WSDL URL	***directory_path*/ThirdFileRead.wsdl**
Partner Link Type	**Read2_plt**
Partner Role	Leave unspecified.
My Role	**Read2_role**

23. Click **OK**.

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.

Creating an Initial Receive Activity

To create an initial receive activity:

1. In the Components window, expand **BPEL Constructs**.
2. Drag a **Receive** activity beneath the **receiveInput** receive activity in the designer.
3. Click the **receive** activity to display its property fields in the Property Inspector or double-click the **receive** icon to display the Receive dialog.

   For information about editing activities in the Property Inspector, see How to Edit BPEL Activities in the Property Inspector.
4. Enter the details described in Table 9-9 to associate the first partner link (**FirstReceive**) with the first receive activity:

Table 9-9 Receive Dialog Fields and Values

Field	Value
Name	`receiveFirst`
Partner Link	**FirstReceive**
Create Instance	Select this check box.

   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** check box, as described in Step 4.
8. Click **OK**.

Creating a Second Receive Activity

To create a second receive activity:

1. From the Components window, drag a second **Receive** activity beneath the **receiveFirst receive** activity.
2. Double-click the **receive** icon to display the Receive dialog.
3. Enter the details described in Table 9-10 to associate the second partner link (**SecondReceive**) with the second receive activity:

Table 9-10 Receive Dialog Fields and Values

Field	Value
Name	`receiveSecond`
Partner Link	**SecondFileRead**
Create Instance	Do *not* select this check box.

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**.

Creating a Third Receive Activity

To create a third receive activity:

1. From the Components window, drag a third **Receive** activity beneath the **receiveSecond receive** activity.
2. Double-click the **receive** icon to display the Receive dialog.
3. Enter the details described in Table 9-11 to associate the third partner link (**ThirdReceive**) with the third receive activity:

Table 9-11 Receive Dialog Fields and Values

Field	Value
Name	`receiveThird`
Partner Link	**ThirdFileRead**
Create Instance	Do *not* select this check box.

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.

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.

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** in each dialog to close the Create Property dialog, the Property Chooser dialog, and the Create Correlation Set dialog.

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** in each dialog to close the Create Property dialog, the Property Chooser dialog, and the Create Correlation Set dialog.

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.

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**.

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.

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.

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.

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**.

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**.

Step 7: Reviewing WSDL File Content

To review WSDL file content:

1. Refresh the Applications window.

   The `NameCorr` and `IDCorr` correlation set properties are defined in the `MyCorrelationSet_Properties.wsdl` file in the Applications window.

   ```
   <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.

   ```
   <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.

```
<import namespace="http://xmlns.oracle.com/MyCorrelationSet"
 location="MyCorrelationSet.wsdl"/>
```

What You May Need to Know About Conversion IDs and Different Composite Revisions

Do not use the same conversion ID for different revisions of a SOA composite application. When correlation sets are used in a BPEL process, you have explicit control over the conversation ID value. Oracle SOA Suite does not interfere or add restrictions on conversation ID value generation. This situation means that even though it appears that Oracle SOA Suite is generating the same conversation ID for different revisions, you actually control this behavior. Oracle SOA Suite does not restrict you from using the same conversation ID for different instances of different revisions.

If you do not use correlation sets, the conversation ID generated is unique and this is not a problem because Oracle SOA Suite decides which conversation ID to generate, and not you.

Oracle SOA Suite does not execute a revision check for callback routing. Routing of callback messages is only based on the following:

- Conversation ID: This is calculated based on the input value and correlation set. If you use the same correlation set for two revisions of processes and enter the same input when creating an instance, both revisions subscribe using the same conversation ID. This causes confusion when a callback for one revision is delivered to another revision.
- Operation name (is the same for both revisions).
- BPEL service component name (is also the same for both revisions).

The concept of a revision number is applicable to Oracle SOA composite applications, and is not part of the BPEL specification. This is why it is not used as part of the routing decision.

There is another complication in which adding a revision as part of callback routing causes problems. When sending a callback, you also specify the endpoint URL. If the endpoint URL does not contain the composite revision (which is extremely likely), the message is assumed to be routed to the default revision. If Oracle SOA Suite runtime adds a revision check as part of callback routing, the callback for the nondefault revision instance is never possible.

For example, assume you have the following BPEL process:

- An entry receive activity named receive_1 (on which a correlation set is used)
- An invoke activity, which invokes a web service
- A receive activity named receive_2

Assume you perform the following steps:

1. Deploy revision 1.0 of composite_A, which includes a BPEL component.
2. Create an instance of revision 1.0, which is using a correlation set, and input a value of `123`, which generates `conv_id = "123"`.

   This process now invokes a web service through a one-way invoke activity and then waits on the receive_2 activity for a callback to arrive.
3. Deploy revision 2.0 of composite_A, which now becomes the default revision.

   A web service sends a callback for the instance for revision 1.0. However, as a part of its URL, it does not specify the revision number. You typically create a callback so that the URL does not use the revision number. This is because web services are external and you cannot change web service settings to continue using a revision tag because it is internal to Oracle SOA Suite and is a concept that the external world does not understand.

   Since a revision number is not specified, the SOA server assumes that the revision number must be 2.0 and, if the routing of the callback takes the revision number into account, it cannot forward this callback intended for 1.0 to the correct revision 1.0. Instead, it attempts to route it to the default revision of 2.0, which does not have any instance waiting for the callback.

   You cannot route callback messages based on revisions. You only receive the option to route callback messages based on the conversion ID (if the correlation set is not used, then even this is not under your control), operation name, and component name.

   For these reasons, different instances must use different conversation IDs (which means different input is used for creating a conversion ID) to avoid confusion, and routing should be solely based on a conversation ID.

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 `fromParts` 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 Mapping WSDL Message Parts in BPEL 2.0.

Routing Messages to the Same Instance

Oracle BPEL Process Manager supports a message aggregation feature. When multiple messages are routed to the same process/partner link/operation name, the first message is routed to create a new instance and subsequent messages can be routed to continue the created instance using a midprocess receive activity.

Message aggregation enables you to use the same operation (or event name) in an entry receive activity and a midprocess receive activity.

Note:

- This feature only performs aggregation, and not resequencing. This feature does not resequence messages arriving out of order into an ordered format. Therefore, the first message only means the first message processed. This may be different from the first message in a time sequence order.
- You must use correlation sets to take advantage of the message aggregation feature.
- Synchronous operations as ambiguous calls (at both beginning and midprocess receive activities) are supported. However, this is not a recommended use of this feature and should be avoided.

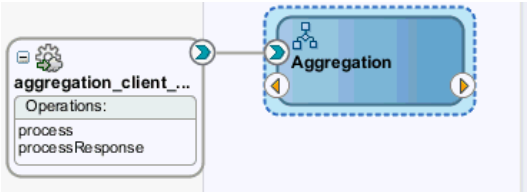
How to Configure BPEL Process Instance Creation

You can control the number of instances to create and use to route messages with the `reenableAggregationOnComplete` property.

To configure BPEL process instance creation:

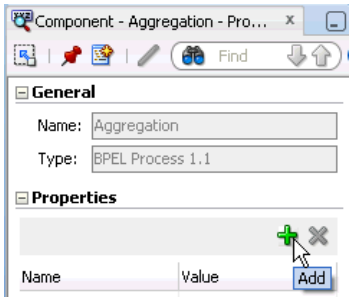
1. In the SOA Composite Editor, select the BPEL process service component, as shown in Figure 9-16.

Figure 9-16 Selected BPEL Process Service Component



2. Go to the Property Inspector in the lower right corner of Oracle JDeveloper. If the Property Inspector is not displayed, select **Property Inspector** from the **View** main menu.
3. In the **Properties** section, click the **Add** icon, as shown in Figure 9-17.

Figure 9-17 Property Inspector



The Create Property dialog is displayed.

4. In the **Name** field, enter the `bpel.config.reenableAggregationOnComplete` deployment descriptor property. The prefix of `bpel.config` is required for this type of deployment descriptor.

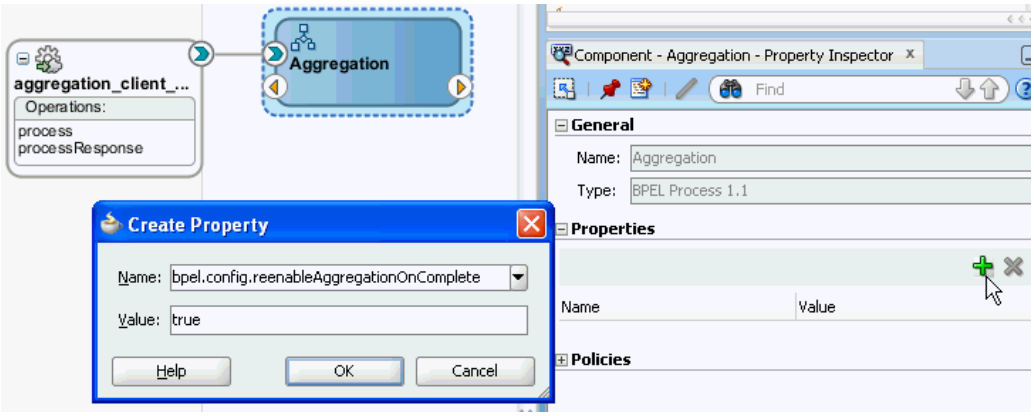
5. In the **Value** field, enter `true`, as described in Table 9-12.

Table 9-12 reenableAggregationOnComplete Property Settings

Value	Description	Example
`true`	Creates a new instance to handle messages. However, there is a window between messages coming in and instance completion. This can result in messages remaining in the `DLV_MESSAGE` table. This setting can result in the occurrence of race conditions. For more information, see Table 9-13.	You invoke messages 1 through 4 for a client using the `initiate` operation. This results in the following actions: • Two instances of the BPEL process are created and completed. • Messages 1 and 2 are routed to the first instance and messages 3 and 4 are routed to the second instance.
`false`	This is the default behavior. This setting causes the aggregation feature to be disabled. Only one instance is created. Messages that are not handled by the instance remain in the `DLV_MESSAGE` table. This setting is recommended for most environments.	You invoke messages 1 through 4 for a client using the `initiate` operation. One instance of the BPEL process is created and completed. Do not attempt to route multiple messages using the same correlation set to one BPEL instance.

Figure 9-18 shows the completed Create Property dialog.

Figure 9-18 Create Property Dialog



6. Click **OK**.

   The `reenableAggregationOnComplete` property with the `bpel.config` prefix looks as follows in the `composite.xml` file.

```
<composite name="Aggregation" revision="1.0" label="2011-07-10_13-52-24_174"
 mode="active" state="on">
. . .
. . .
<component name="Aggregation" version="1.1">
    <implementation.bpel src="Aggregation.bpel"/>
      <property name="bpel.config.reenableAggregationOnComplete" type="xs:string"
              many="false" override="may">true</property>
  </component>
. . .
. . .
</composite>
```

How to Use the Same Operation in Entry and Midprocess Receive Activities

Assume you create a correlation set as shown in the example that follows. All messages to Oracle BPEL Process Manager are routed to the same operation name. The messages have the same correlation ID. The interface WSDL does not differentiate between the entry activity (`receiveInput`) and the midprocess receive activity (`Continue_Receive`). All messages are processed using the `initiate` operation. A single instance is created to which to route all messages.

This differs from releases before 11*g* Release 1 11.1.1.6, in which you needed to define different operation names on the same partner link. The process had to expose two operations and the caller had to choose the correct operation name.

```
<receive name="receiveInput" partnerLink="client" portType="client:BPELProcess1"
 operation="initiate" variable="inputVariable" createInstance="yes">
 <correlations>
  <correlation initiate="yes" set="CorrelationSet_1"/>
 </correlations>
</receive>

<!-- Asynchronous callback to the requester. (Note: the callback location and
 correlation id is transparently handled using WS-addressing.) -->
<assign name="Assign_1">
 <copy>
  <from variable="inputVariable" part="payload"
 query="/client:BPELProcess1ProcessRequest/client:input"/>
```

```
  <to variable="Invoke_1_initiate_InputVariable" part="payload"
 query="/ns1:BPELProcess2ProcessRequest/ns1:input"/>
 </copy>
</assign>

<receive name="Continue_Receive" partnerLink="client"
 portType="client:BPELProcess1" operation="initiate" variable="inputVariable"
 createInstance="no">
 <correlations>
  <correlation initiate="no" set="CorrelationSet_1"/>
 </correlations>
</receive>
```

For event delivery network (EDN) business events, you substitute the `operation` attribute with `bpelx:eventName` in both the entry and midprocess receive activities.

```
bpelx:eventName="ns3:initiateEvent"/>
```

Information is maintained in the `DLV_AGGREGATION` table:

- Conversation ID
- Domain name
- Component name and type
- Composite name, label, and revision
- State
- Received date
- CI key
- Primary key

This information can be deleted from this table with the purge scripts or from the Auto Purge page in Oracle Enterprise Manager Fusion Middleware Control. For more information about both of these options, see the *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

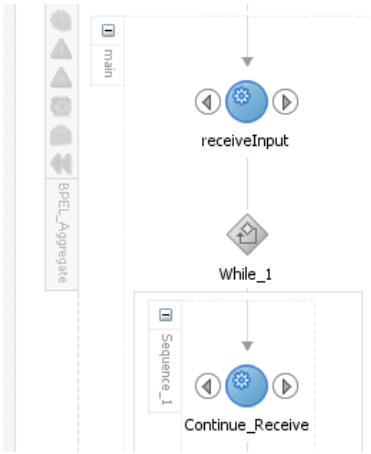
How to Route a Message to a New or Existing Instance when Using Correlation Sets

For a BPEL process using correlation sets, the correct routing is performed. The message can be either of the following:

- An invoke message creating a new instance
- A callback message continuing an existing instance

Figure 9-19 shows entry and midprocess receive activities using the same operation (`process`).

Figure 9-19 Routing a New Message to a New or Existing Instance



The following provides an example of the entry and midprocess receive activities using the same operation (`process`).

```
<receive name="receiveInput" partnerLink="client" portType="client:BPELProcess1"
 operation="process" variable="inputVariable" createInstance="yes">
 <correlations>
  <correlation initiate="yes" set="CorrelationSet_1"/>
 </correlations>
</receive>

<!-- some business logic -->

<while name="While_1" condition=*loop for 3 iterations*>
 <sequence name="Sequence_1">
  <receive name="Continue_Receive" partnerLink="client"
 portType="client:BPELProcess1" operation="process" variable="inputVariable"
 createInstance="no">
   <correlations>
    <correlation initiate="no" set="CorrelationSet_1"/>
   </correlations>
  </receive>

<!-- some business logic -->

 </sequence>
</while>
```

In the initial scenario in the preceding example, the following actions occur in BPEL process P1:

- A partner provides four messages (message 1, message 2, message 3, and message 4) for the same partner (correlation ID `101`).
- Message 1 creates a new instance of BPEL process P1. This message is marked as an invoke message.
- Messages 2, 3, and 4 are received using the `Continue_Receive` activity. These messages are marked as callback messages.
- The instance closes because three iterations of the `while` loop are expected.

Assume now that additional messages are routed, which can potentially cause race conditions to occur. Table 9-13 provides details.

Table 9-13 Message Delivery Scenarios

Scenario	Description	Marked as Invoke Message	Marked as Callback Message
1	Assume the partner now provides message 5 for the same correlation ID (`101`). Message 5 creates a new instance of BPEL process P1 and waits on the `Continue_Receive` activity inside the `while` loop for three more messages (6, 7, and 8).	• Message 1 • Message 5	• Message 2 • Message 3 • Message 4 • Message 6 • Message 7 • Message 8
2	If messages 4 and 5 are received within a small time window, it is possible that message 4 is closing the instance BPEL process P1 and message 5 is routed as a callback to that instance. This scenario can cause a race condition. For example: • When message 6 arrives, it is routed to the entry receive activity of the new instance. • Messages 7 and 8 are routed to the `Continue_Receive` activity. • Message 5 is routed to the `Continue_Receive` activity only by the recovery part of the BPEL process service engine. This is because it initially was routed to a closed instance and could not be handled.	• Message 1 • Message 6	• Message 2 • Message 3 • Message 4 • Message 5 • Message 7 • Message 8
3	This is similar to scenario 2. However, in this case, messages 7, 8, and 9 are not received. For example: • Message 5 becomes an unhandled callback message waiting for a subscriber. • BPEL process service engine recovery tries to process message 5 and fails because there is no subscriber available. There are several options for message recovery. • Limit recovery of callback messages with the System MBean Browser property **maxRecoverAttempt** in Oracle Enterprise Manager Fusion Middleware Control. This count specifies the number of attempts made by automatic recovery to recover an invoke/callback message. Once the number of recover attempts exceeds this count, the state of the message is changed to exhausted. For more information, see Section "Configuring Automatic Recovery Attempts for Invoke and Callback Messages" in *Administering Oracle SOA Suite and Oracle Business Process Management Suite*. • Write a custom SQL script to check that the `criteriaCallback` has `state` set to `0`. The correlation value for this callback exists in `CORRELATION_GROUP` in a closed state (`state` = `0`). This indicates that the callback message is marked for a closed aggregation instance. You can cancel/purge these instances based on business logic. **Note**: BPEL is designed as a conversation-based system. At any point in which unsolicited messages are not being handled, the application is always aware of the messages coming as part of correlation aggregation and chooses to subscribe and process or ignore the message as required by business needs.	• Message 1 • Message 6	• Message 2 • Message 3 • Message 4 • Message 5

10

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 flows are especially useful when you must perform several time-consuming and independent tasks. This chapter also describes how to customize the number of parallel branches.

This chapter includes the following sections:

Introduction to Parallel Flows in BPEL Processes

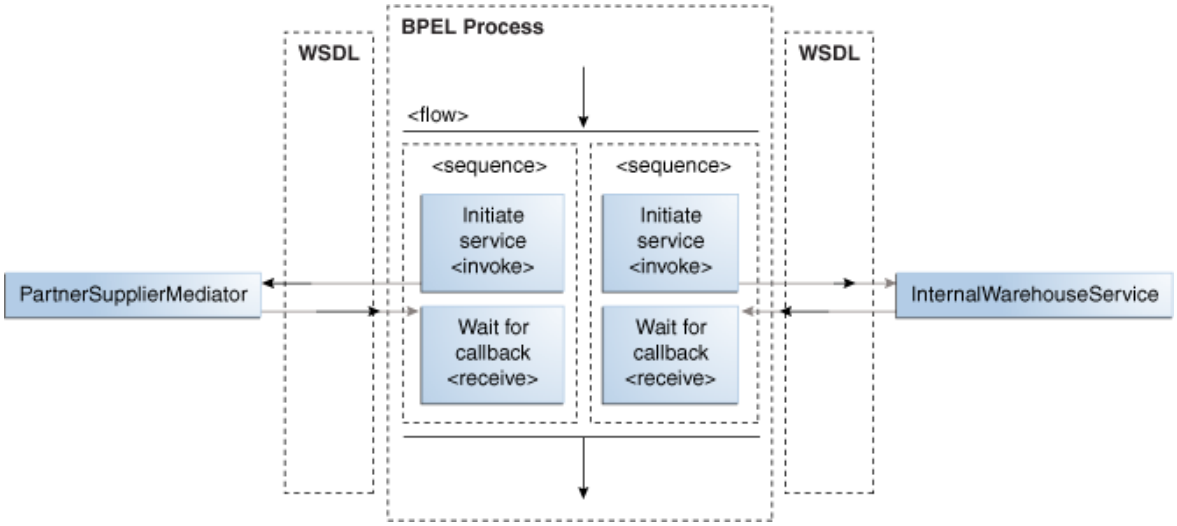
A BPEL process service component must sometimes gather information from multiple asynchronous sources. Because each callback can take an undefined amount of time (hours or days), it may take too long to call each service one at a time. By breaking the calls into a parallel flow, a BPEL process service component can invoke multiple web services at the same time, and receive the responses as they come in. This method is much more time efficient.

Figure 10-1 shows a flow activity named Retrieve_QuotesFromSuppliers. The Retrieve_QuotesFromSuppliers flow activity sends order information to two suppliers in parallel:

- An internal warehouse (InternalWarehouseService)
- An external partner warehouse (PartnerSupplierMediator)

The two warehouses return their bids for the order to the flow activity. Here, two asynchronous callbacks execute in parallel. One callback does not have to wait for the other to complete first. Each response is stored in a different global variable.

Figure 10-1 Parallel Flow Invocation



What You May Need to Know About the Execution of Parallel Flow Branches in a Single Thread

Branches in flow, flowN, and forEach activities are executed serially in a single thread (that is, the *N*th branch is executed only after *N*-1 execution has completed). Execution is not completely parallel. This is because the branches do not execute in concurrent threads in this mode. Instead, one thread starts executing a flow branch until it reaches a blocking activity (for example, an synchronous invoke). At this point, a new thread is created that starts executing the other branch, and the process continues. This creates the impression that the flow branches are executing in parallel. In this mode, however, if the flow branches do not define a blocking activity, the branches still execute serially.

This design is intended for several reasons:

- To prevent you from accidentally spawning too many threads and overloading the system, single threading is the default method. However, you can tune threads in other places, such as adapter polling threads, BPEL process service engine threads, and Oracle WebLogic Server work managers.
- The BPEL process specification does not provide a mechanism to ensure the thread safety of BPEL variables (that is, a lack of a synchronized qualifier such as in Java), which is necessary for true multithreaded programming.
- The implication of transaction rollbacks in one of the branches is undefined.

To achieve pseudo-parallelism, you can configure invoke activities to be nonblocking with the `nonBlockingInvoke` deployment descriptor property. When this property is set to `true`, the process manager creates a new thread to perform each branch's invoke activity in parallel.

For more information about the `nonBlockingInvoke` property, see How to Define Deployment Descriptor Properties in the Property Inspector.

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.

Note:

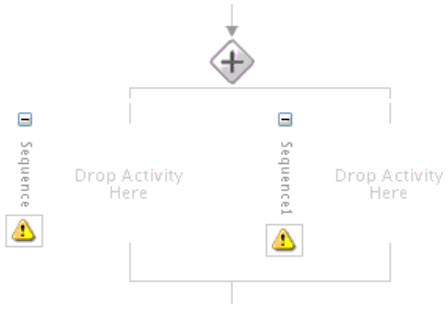
Branches in a flow activity are executed serially in a single thread. For more information, see What You May Need to Know About the Execution of Parallel Flow Branches in a Single Thread.

How to Create a Parallel Flow

To create a parallel flow:

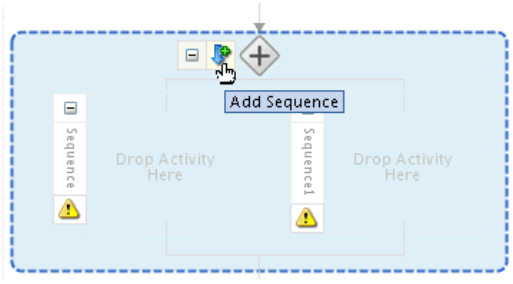
1. In the Components window, expand **BPEL Constructs** > **Structured Activities**.
2. Drag a **Flow** activity into the designer.
3. Click the **+** sign to expand the flow activity, as shown in Figure 10-2.

Figure 10-2 Flow Activity



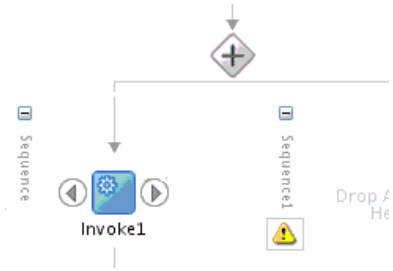
The flow activity initially includes two branches, each with a box for functional elements. Populate these boxes as you do a scope activity, either by building a function or dragging activities into the boxes. You can add additional branches by highlighting the flow activity and clicking the **Add Sequence** icon. Figure 10-3 provides details.

Figure 10-3 Add Sequence Icon



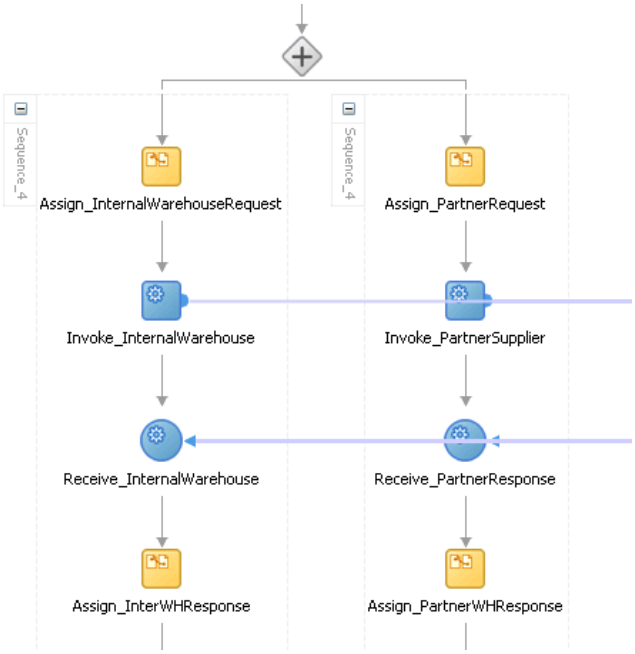
4. Drag and define additional activities on each side of the flow to invoke multiple services at the same time. Figure 10-4 provides details.

Figure 10-4 Expanded Flow Activity



When complete, flow activity design can look as shown in Figure 10-5. This example shows the **Retrieve_QuotesFromSuppliers** flow activity. Two branches are defined for receiving bids: one for **InternalWarehouseService** and the other for **PartnerSupplierMediator**.

Figure 10-5 Flow Activity After Design Completion



What Happens When You Create a Parallel Flow

A flow activity typically contains many sequence activities. Each sequence is performed in parallel. The following example 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 the following example, each sequence in the flow contains assign, invoke, and receive activities.

```
<flow name="Retrieve_QuotesFromSuppliers">
    <sequence name="Sequence_4">
        <assign name="Assign_InternalWarehouseRequest">
           <copy>
           <from>$inputVariable.gOrderInfoVariable/ns3:CardNum</from>
           <to>lInternalWarehouseInputVariable/ns4:ccnb</to>
         </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>
```

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.

The following example 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.

```
<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>
```

The preceding code 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> and <sources>.

Table 10-1 provides details.

Table 10-1 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>
```

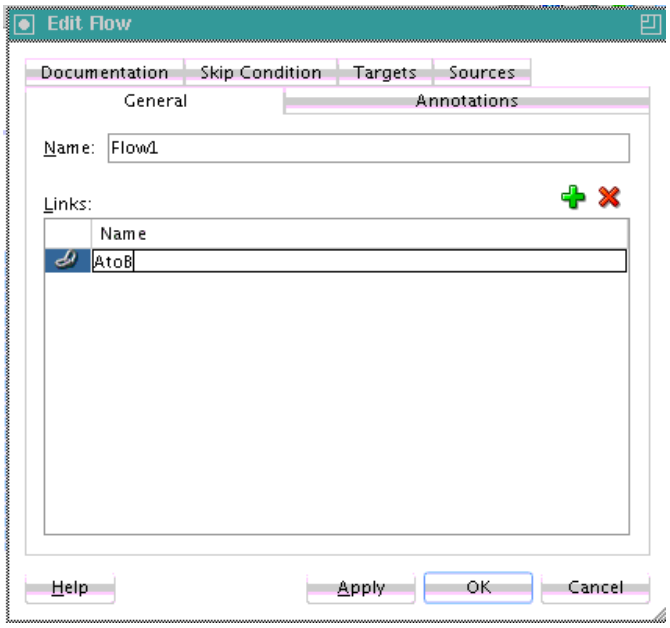
How to Create Synchronization Between Activities Within a Flow Activity

To create synchronization between activities within a flow activity:

> **Note:**
>
> The **Sources** and **Targets** tabs are only available in BPEL 2.0 projects. For BPEL 1.1 projects, you must directly edit the BPEL file to use this functionality.

1. Create a flow activity. For information, see 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 10-6.

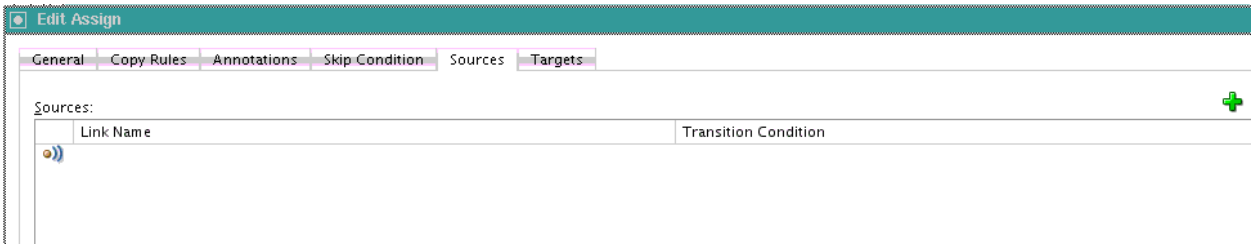
Figure 10-6 Link Name Creation



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 10-7.

Figure 10-7 Source Activity



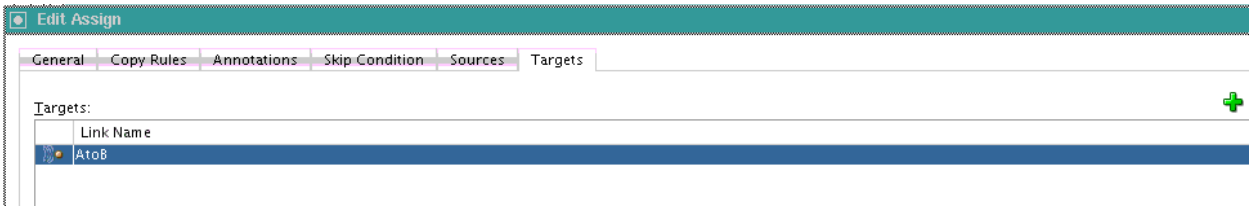
Each source activity can specify an optional **Transition Condition** as a safe guard for following the specified link. Click the row in this column to invoke the **Browser** icon for accessing the Expression Builder dialog for creating a condition. If the **Transition Condition** column is left blank, 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 10-8.

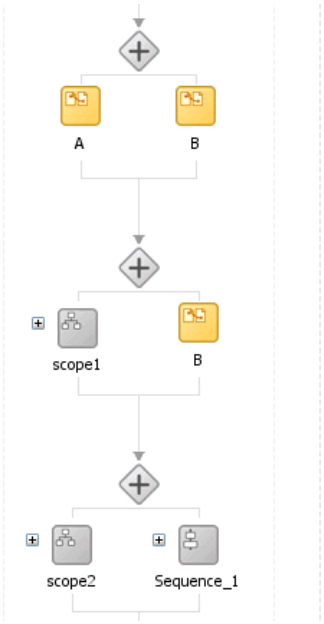
Figure 10-8 Target Activity



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 10-9.

Figure 10-9 Three Flow Activities Synchronized with Links



What Happens When You Create Synchronization Between Activities Within a Flow Activity

The following example 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.

```
<!-- 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>
```

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.

The following provides an example of a join condition.

```
<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>
```

Customizing the Number of Parallel Branches

This section describes how to customize the number of parallel branches with the following activities:

- A forEach activity in a BPEL version 2.0 project
- A flowN activity in a BPEL version 1.1 project

Note:

Branches in flowN and forEach activities are executed serially in a single thread. For more information, see What You May Need to Know About the Execution of Parallel Flow Branches in a Single Thread.

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** check box 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.

The following example shows the forEach activity syntax.

```
<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.

How to Create a forEach Activity

To create a forEach activity:

1. In the Components window, expand **BPEL Constructs** > **Structured Activities**.
2. Drag a **For Each** activity into the designer, as shown in Figure 10-10.

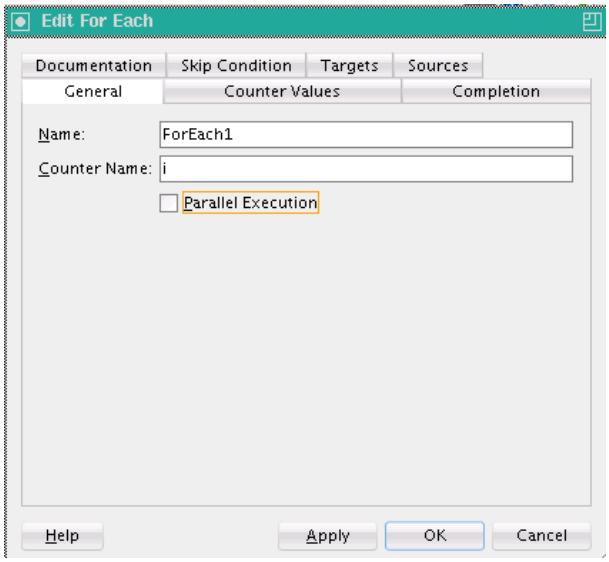
Note the contained scope activity in the forEach activity.

Figure 10-10 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 10-11.

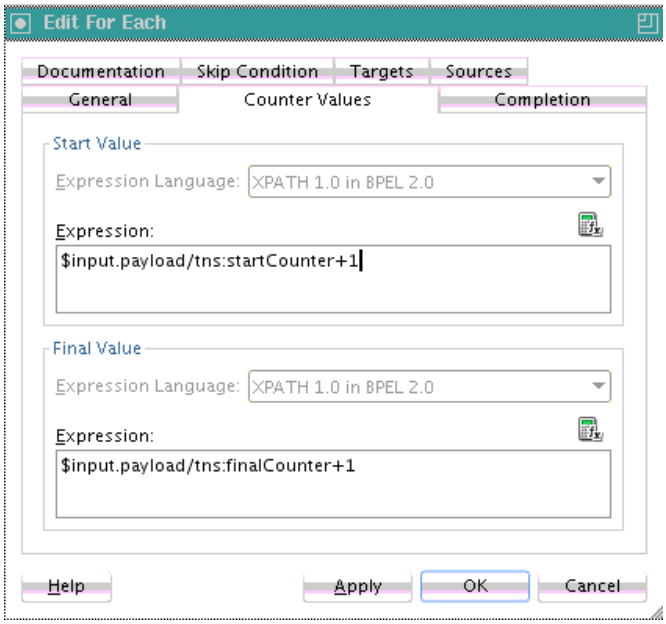
   If the **Parallel Execution** check box is selected, all looping iterations are started at the same time and processed in parallel. The next branch starts even if the previous branch has not completed. If not selected, the next branch does not start until the previous branch has completed.

Figure 10-11 General Tab of the forEach Activity



5. Click the **Counter Values** tab.
6. Click the **Expression Builder** icon to enter the starting counter value and final counter value, as shown in Figure 10-12.

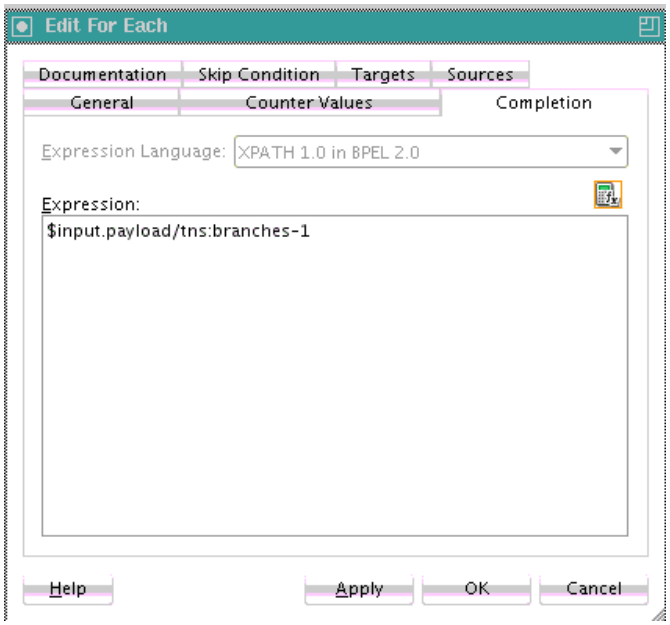
Figure 10-12 Counter Values Tab of the forEach Activity



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 above the **Expression** field to enter a condition. Figure 10-13 provides details.

Figure 10-13 Completion Tab of the forEach Activity



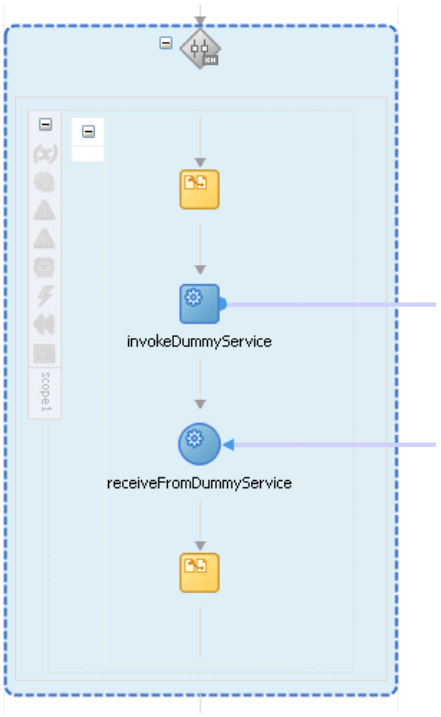
9. Click **Apply**, then **OK**.

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 10-14.

Figure 10-14 forEach Activity with Contained and Expanded Scope Activity



What Happens When You Create a forEach Activity

The following example shows the `.bpel` file after design is complete for a sequential `forEach` activity.

```
<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 requester -->
    <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 requester -->
  <reply name="replyOutput" partnerLink="client"
      portType="tns:Test" operation="process" variable="output"/>
</sequence>
```

The following example shows the `.bpel` file after design is complete for a parallel `forEach` activity.

```
<sequence>
    <!-- pick input from requester -->
    <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 requester -->
<reply name="replyOutput" partnerLink="client"
    portType="tns:Test" operation="process" variable="output"/>
</sequence>
```

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.

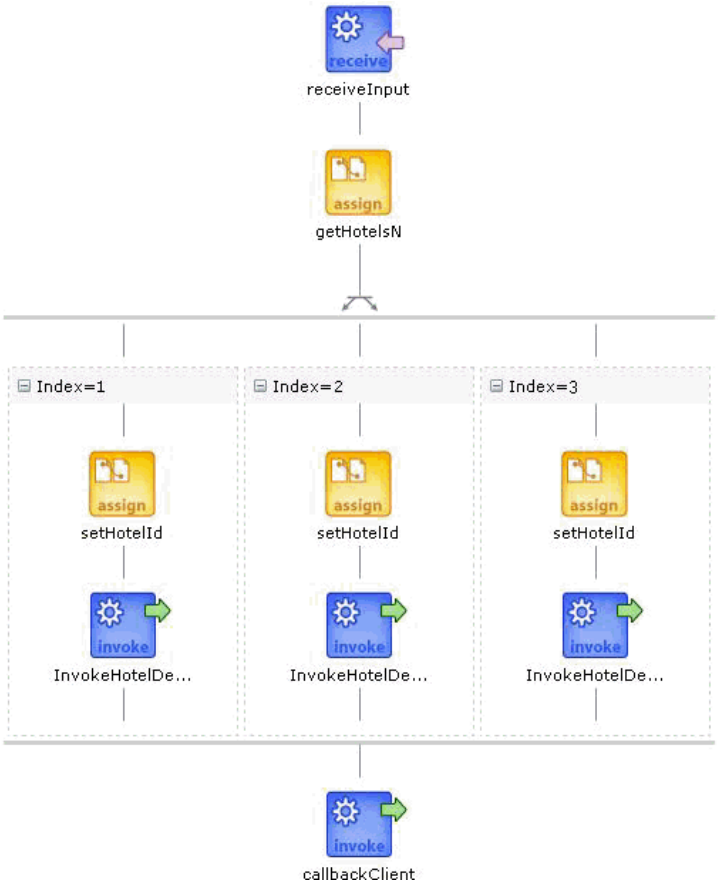
For example, suppose there is an array of data. The BPEL process service component uses a `count` function to determine the number of elements in the array. The process then sets `N` to be the number of elements. The index variable starts at a preset value (zero is the default), and flowN creates branches 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`. The flowN activity terminates when the index variable reaches the value of `N`. For example, if the array contains `3` elements, `N` is set to `3`. Assuming the index

variable begins at 1, the flowN activity creates three parallel branches with indexes 1, 2, and 3.

The flowN activity can use data from other sources as well, including data obtained from web services.

Figure 10-15 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 10-15 Oracle Enterprise Manager Fusion Middleware Control View of the Execution of a flowN activity



How to Create a flowN Activity

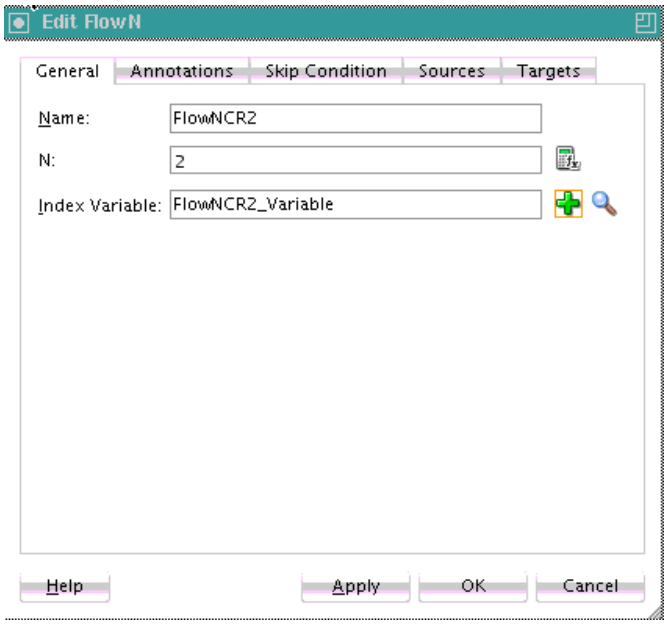
To create a flowN activity:

1. In the Components window, expand **Oracle Extensions**.
2. Drag a **FlowN** activity into the designer.
3. Click the **+** sign to expand the **FlowN** activity.
4. Click the **FlowN** activity to display its property fields in the Property Inspector or double-click the **FlowN** activity.

For information about editing activities in the Property Inspector, see How to Edit BPEL Activities in the Property Inspector.

Figure 10-16 shows the FlowN dialog.

Figure 10-16 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 10-17 shows how a FlowN activity appears with additional activities.

Figure 10-17 FlowN Activity with Additional Activities

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 example shows the 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"
    -->
```

The following actions take place. 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. The following provides an example:

```
    <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. The following provides an example:

```
<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 the following example then uses the index variable to concatenate the hotel entries into a list:

```
<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. The following provides an example:

```
<!-- 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"/>
     </sequence>
   </bpelx:flowN>
```

Finally, the BPEL process sends detailed information on each hotel to the client partner link. The following provides an example:

```
   <invoke name="callbackClient" partnerLink="client"
portType="client:NflowHotelsCallback" operation="onResult"
inputVariable="outputVariable"/>
 </sequence>
 </sequence>
```

11

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 also describes how to use the switch, if, while, and repeatUntil activities to define conditional branching and specify XPath expressions that enable you to bypass execution of activities.

This chapter includes the following sections:

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:

- 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 Defining Conditional Branching with the If Activity in BPEL 2.0.

- 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 Defining Conditional Branching with the Switch Activity in BPEL 1.1 .

- While activity

Enables you to create a while loop to select between two actions. Defining Conditional Branching with the While Activity 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. Using Events and Timeouts in BPEL Processes explains this feature in detail.

Note:

You can also define conditional branching logic with business rules. See *Designing Business Rules with Oracle Business Process Management*.

Defining Conditional Branching with the If or Switch Activity

This section describes how to define conditional branching with the following activities:

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

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.

The following example shows the if activity syntax:

```
<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>
```

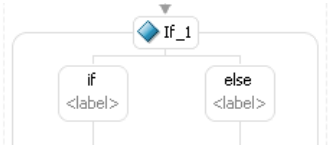
How to Create an If Activity

To create an If activity:

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

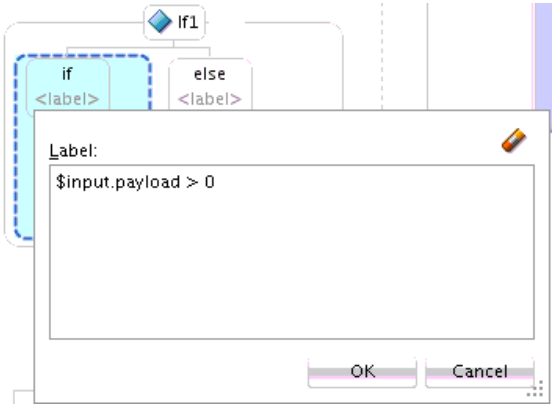
   The **if** and **else** conditions are displayed, as shown in Figure 11-1.

   Figure 11-1 If Activity

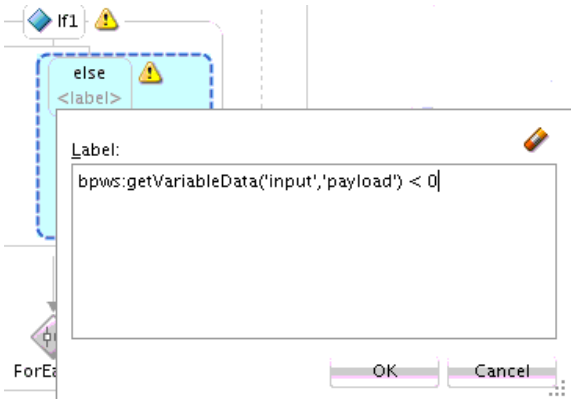
3. Click the **if** branch.
4. In the **Condition** field, enter a condition, as shown in Figure 11-2. You can also click the **XPath Expression Builder** icon to invoke the Expression Builder dialog.

   Figure 11-2 if Branch of the If Activity

5. Click **OK**.
6. Drag and define additional activities into the **if** condition, as needed. These activities are executed if the if condition evaluates to true.
7. Click the **elseif** branch (if you added this branch).
8. In the **Condition** field, enter a condition, as shown in Figure 11-3.

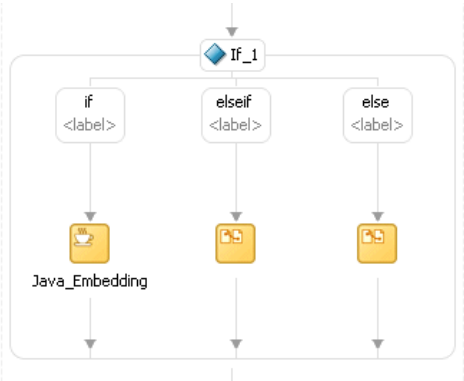
Figure 11-3 elseif Branch of the If Activity



9. Click **OK**.
10. If you want to add elseif conditions, highlight the **If** activity, and select the **Add** icon to invoke a menu.
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 11-4 shows a completed if activity in which each branch includes contained activities.

Figure 11-4 Completed If Activity



What Happens When You Create an If Activity

The following code 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.

```
<sequence>
  <!-- receive input from requester -->
    <receive name="receiveInput" partnerLink="client" portType="tns:Test"
        operation="process" variable="input" createInstance="yes"/>
    <!-- assign default value -->
```

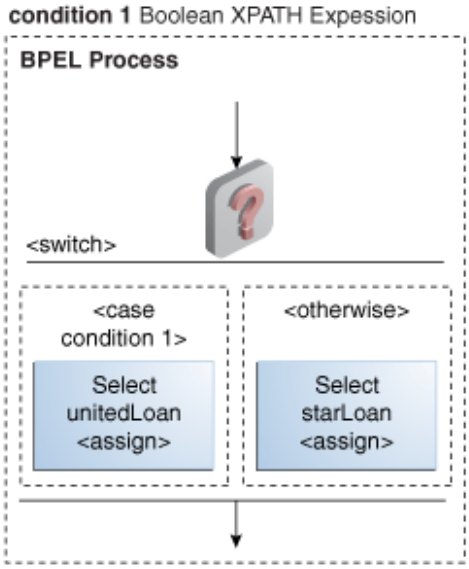
```
<assign>
  <copy>
    <from>'Value is greater than zero'</from>
    <to>$output.payload</to>
  </copy>
<assign>
  <copy>
    <from>'Value is greater than zero'</from>
    <to>$output.payload</to>
  </copy>
</assign>
<!-- switch depends on the input value field -->
<if>
  <condition>$input.payload > 0</condition>
  <extensionActivity>
    <bpelx:exec name="Java_Embedding" version="1.5" language="java">
      System.out.println("if condition is true.\n");
    </bpelx:exec>
  </extensionActivity>
  <elseif>
    <condition>bpws:getVariableData('input', 'payload') &lt; 0</condition>
    <assign>
      <copy>
        <from>'Value is less than zero'</from>
        <to>$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 requester -->
<reply name="replyOutput" partnerLink="client"
   portType="tns:Test" operation="process" variable="output"/>
</sequence>
```

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 11-5 provides an overview of a BPEL conditional branching process that has been defined in a switch activity.

Figure 11-5 Conditional Branching



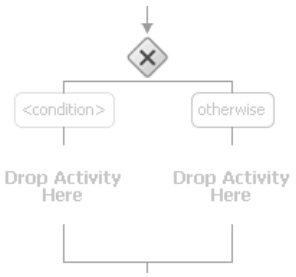
How to Create a Switch Activity

To create a switch activity:

1. In the Components window, expand **BPEL Constructs**.
2. Drag a **Switch** activity into the designer, as shown in Figure 11-6.

   The **Switch** activity has two switch case branches by default, each with a box for functional elements. If you want to add more branches, select the entire switch activity, right-click, and select **Add Switch Case** from the menu.

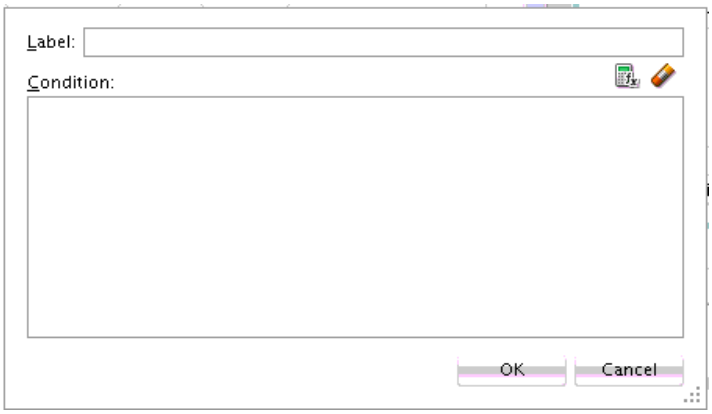
   Figure 11-6 Switch Activity

3. In the first branch, double-click the **condition** box.

   A dialog for entering a condition is displayed, as shown in Figure 11-7.

Figure 11-7 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 **Condition** field, click the **Expression Builder** icon to access the Expression Builder dialog.
6. Create your expression.

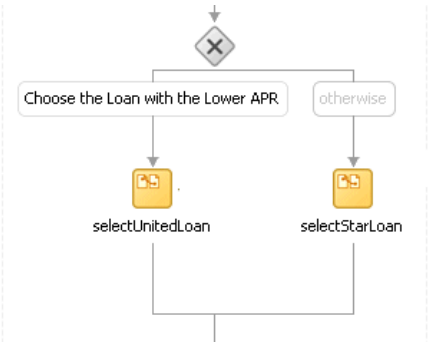
   ```
   bpws:getVariableDate('loanOffer1','payload','/loanOffer/APR') >
   bpws:getVariableData('loanOffer2','payload','/loanOffer/APR')
   ```

   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.
7. Click **OK**.

   The expression is displayed. The value you entered in the **Label** field of the dialog becomes the name of the condition branch.
8. Click **OK**.
9. Add and configure additional activities as needed. Figure 11-8 provides details.

Figure 11-8 Switch Activity Design



What Happens When You Create a Switch Activity

A switch activity, such as a flow activity, has multiple branches. In the example that follows, 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, as needed.

```
<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>
```

Defining Conditional Branching with the While Activity

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.

How To Create a While Activity

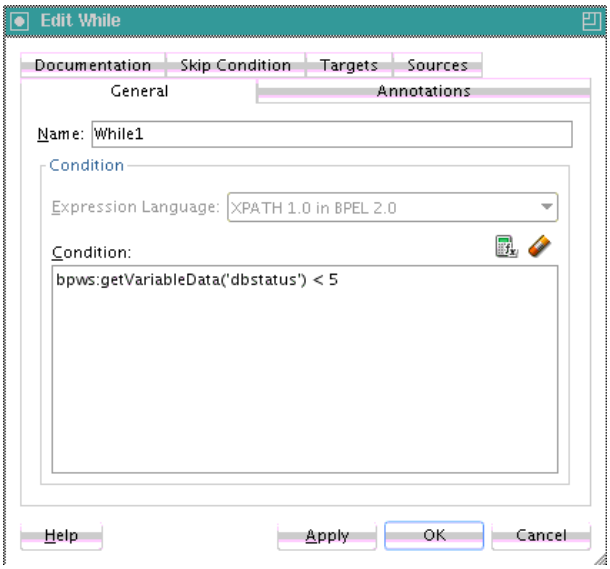
To create a while activity:

1. In the Components window, 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. Click the **XPath Expression Builder** icon to open the Expression Builder dialog.
6. Enter an expression to perform repeatedly, as shown in Figure 11-9. This action is performed until the given boolean while condition is no longer true. In this example, this activity is set to loop while less than 5.

Figure 11-9 While Activity with an Expression



7. Click **OK** when complete.

What Happens When You Create a While Activity

The code that follows provides an example of the `.bpel` file after design completion. The while activity includes a scope activity. The scope activity includes sequence and fault handlers at the top level. The sequence includes invoke and assign activities and fault handlers that define a `catchAll` containing assign and wait activities wrapped in a sequence.

The following code calls an external service. If the external service throws a fault, the fault handler catches the fault and increments the `dbStatus` variable value.

Therefore, the exit condition of the while loop is either of the following:

- There is no exception, upon which the `dbStatus` value is set to a value of `10`, which results in the while condition evaluating to false.
- After throwing a fault five times, the `dbStatus` value is `5`, and the while condition returns false.

```
<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>
```

Note:

The while activity code fragment in the preceding example uses a BPEL 1.1 construct of `bpws:getVariableData('dbStatus')`. For BPEL 2.0, variables are referenced directly using `$` sign and dot (`.`) notation. For example:

```
<while name="While1">
   <condition>$inputVariable.payload/client:counter > 0
   </condition>
```

Defining Conditional Branching with the repeatUntil Activity

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.

How to Create a repeatUntil Activity

To create a repeatUntil activity:

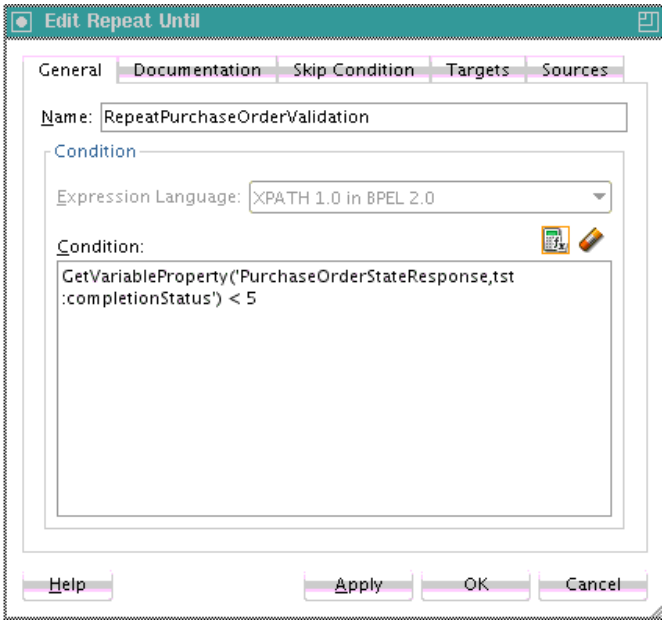
1. In the Components window, expand **BPEL Constructs**.
2. Drag a **Repeat Until** activity into the designer.
3. Click the **Repeat Until** activity to display its property fields in the Property Inspector or double-click the **Repeat Until** activity.

   For information about editing activities in the Property Inspector, see How to Edit BPEL Activities in the Property Inspector.
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**.

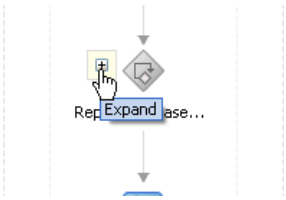
   The condition you entered is displayed in the Repeat Until dialog, as shown in Figure 11-10.

Figure 11-10 Completed Repeat Until Dialog



7. Click **Apply**, then **OK**.
8. Expand the **Repeat Until** activity, as shown in Figure 11-11.

Figure 11-11 repeatUntil Activity Being Expanded



9. Design the body of the activity by dragging in activities from the Components window and defining their property values. These activities are evaluated until the XPath expression condition is evaluated to true.

What Happens When You Create a repeatUntil Activity

The following 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`.

```
<repeatUntil>
   <sequence>
      <invoke name="PurchaseOrderValidation" ... />
      <receive name="receiveValidation"
         partnerLink="PurchaseOrderValidation"
         operation="returnPurchaseOrderValidation"
         variable="PurchaseOrderStatusResponse" />
   </sequence>
   <condition>
      bpel:getVariableProperty(
      "PurchaseOrderStatusResponse","tst:completionStatus") < 5
   </condition>
</repeatUntil>
```

Specifying XPath Expressions to Bypass Activity Execution

Oracle provides an extension that enables you to specify an XPath expression in an activity in BPEL versions 1.1 and 2.0 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"/>
```

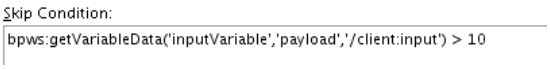
The `bpelx:skipCondition` attribute causes an XPath expression to be evaluated immediately upon creation of the activity instance. If the skip expression returns a false boolean value, the activity is executed. If the skip expression returns a true boolean value, the activity is completed immediately and execution moves to the activity immediately following that one.

How to Specify XPath Expressions to Bypass Activity Execution

To specify XPath expressions to bypass activity execution:

1. In the Components window, 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 11-12 provides details.

   Figure 11-12 Skip Condition XPath Expression

5. Click **Apply**, then **OK**.

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 the following code, when evaluated to true (for example, `input` is `20`), causes the assign activity to be skipped.

```
<sequence name="main">
. . .
. . .
<assign name="Assign_1"

bpelx:skipCondition="number(bpws:getVariableData('inputVariable','payload','/client:
 process/client:input')) > 10">
    <copy>
       <from expression="'Assign Block is not Skipped'"/>
          <to variable="inputVariable" part="payload"
             query="/client:process/client:input"/>
    </copy>
</assign>
. . .
```

```
. . .
</sequence>
```

The `bpelx:skipCondition` attribute is equivalent to a switch/case structured activity with a single case element with a condition that is the opposite of the skip condition.

The following example shows the `bpelx:skipCondition` attribute in BPEL 1.1. 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.

```
<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 the following example.

```
<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>
```

In BPEL 2.0, the `bpelx:skipCondition` syntax appears as a child element of an activity. The following code provides an example of an assign activity with this convention.

```
<assign name="Assign4">
<bpelx:skipCondition>ora:getNodeValue($inputVariable.payload/client:input) > 5
</bpelx:skipCondition><copy>
            <from>"dummy result"</from>
            <to>$outputVariable.payload/client:result</to>
          </copy></assign>
```

You can also use built-in and custom XPath functions within the skip condition expression. The following code provides several examples.

```
<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.

12

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. This chapter also describes how to use the fault management framework to catch faults and perform user-specified actions defined in a fault policy file.

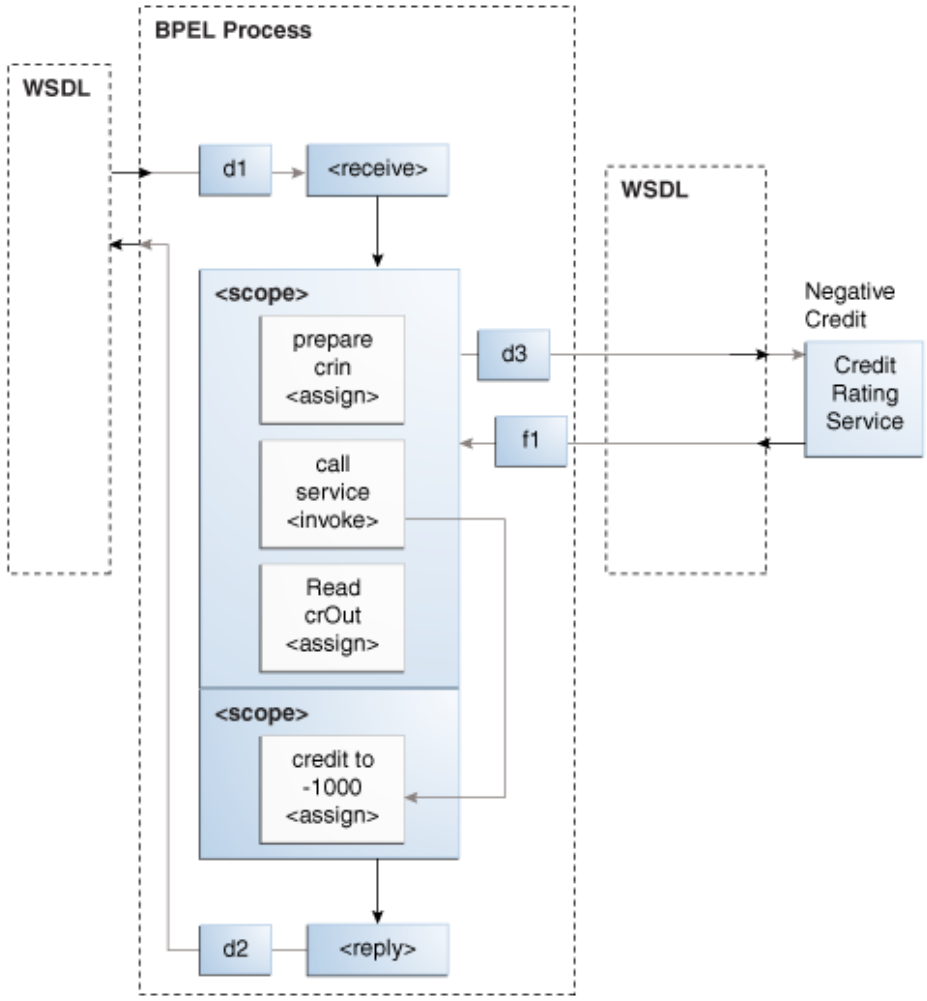
This chapter includes the following sections:

Introduction to a Fault Handler

Fault handlers define how the BPEL process service component responds when target services return data other than what is normally expected (for example, returning an error message instead of a number). An example of a fault handler is where the web service normally returns a credit rating number, but instead returns a negative credit message.

Figure 12-1 provides an example of how a fault handler sets a credit rating variable to `-1000`.

Figure 12-1 Fault Handling



The code segment in the following example defines the fault handler for this operation in the BPEL file:

```
<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.

Introduction to BPEL Standard Faults

This section identifies the standard faults for BPEL 1.1 and BPEL 2.0.

BPEL 1.1 Standard Faults

This section identifies the standard faults for BPEL 1.1. Unless otherwise noted below, the 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`
- `assertFailure`
- `coordinationFault`
- `entityInternalNestedError`
- `maxLoopCountExceeded`
- `owsmPolicyFault`
- `rollback`
- `timeout`

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">
```

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`

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.
  - 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` 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.

Introduction to the Business and Runtime Fault 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

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 file. A business fault can be caught with a `faultHandler` using the `faultName` and a `faultVariable`.

```
<catch faultName="ns1:faultName" faultVariable="varName">
```

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 for a variety of reasons, including the following:

- 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.

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 the following example defines the `messageType`:

```
<?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.

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 retriable. This type of fault usually must be fixed by human intervention.

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.

replayFault

A `replayFault` replays the activity inside a scope. At any point inside a scope, this fault is migrated up to the scope. These faults are not populated into a common fault, but are an indication to BPEL to re-execute the scope. The server then re-executes the scope from the beginning.

How to Add and Propagate Fault Handling in a Synchronous BPEL Process

This section describes how to add and propagate fault handling in a synchronous BPEL process. During the design, you perform the following tasks:

- Modify the existing schema and WSDL files to include fault element, fault message, and fault operation details.
- Add fault handling to the BPEL process (specifically, a catch activity).
- Create a fault variable with the fault message type you specified in the WSDL file.
- Add assign and reply activities with additional fault handling details.

Edit the Schema and WSDL Files

To edit the schema and WSDL files:

1. Create a synchronous BPEL process (for this example, named `TestProcess`) using the default settings in the Create BPEL Process dialog.
2. In the **Schemas** folder of the Applications window, double-click the **TestProcess.xsd** file.
3. Click **Source** view, and add a new element called `processFault`:

```
<element name="processFault">
     <complexType>
         <sequence>
              <element name="result" type="string"/>
         </sequence>
     </complexType>
</element>
```

4. In the Applications window, expand the **WSDLs** folder.
5. Double-click the **TestProcess.wsdl** file.
6. Click **Source** view, and add a new message type called `TestProcessFaultMessage`.

```
<wsdl:message name="TestProcessFaultMessage">
     <wsdl:part name="payload" element="client:processFault"/>
</wsdl:message>
```

7. Edit the `operation` element in the WSDL file to add a fault.

```
<wsdl:operation name="process">
      <wsdl:input  message="client:TestProcessRequestMessage" />
      <wsdl:output message="client:TestProcessResponseMessage"/>
      <wsdl:fault name="FaultResponse" message="
       client:TestProcessFaultMessage"/>
</wsdl:operation>
```

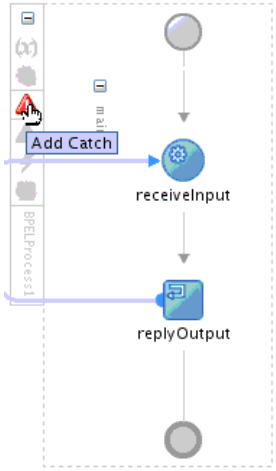
8. From the **File** menu, select **Save**.

Add a Fault Handler

To add a fault handler:

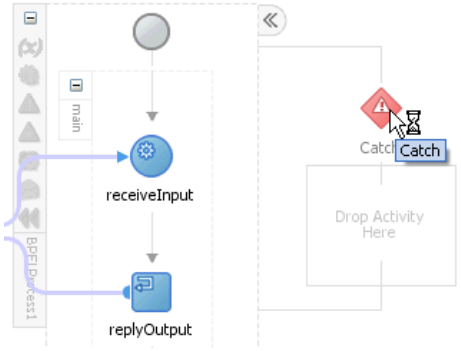
1. In the Applications window, expand **SOA** > **BPEL**.
2. Double-click **TestProcess.bpel**.
3. Click the **Add Catch** icon in the BPEL process to add a catch activity as the fault handler for the BPEL process. You can also use a CatchAll activity. Figure 12-2 provides details.

Figure 12-2 Add Catch Icon



4. Double-click the catch activity to specify the system fault. Figure 12-3 provides details.

Figure 12-3 Catch Activity



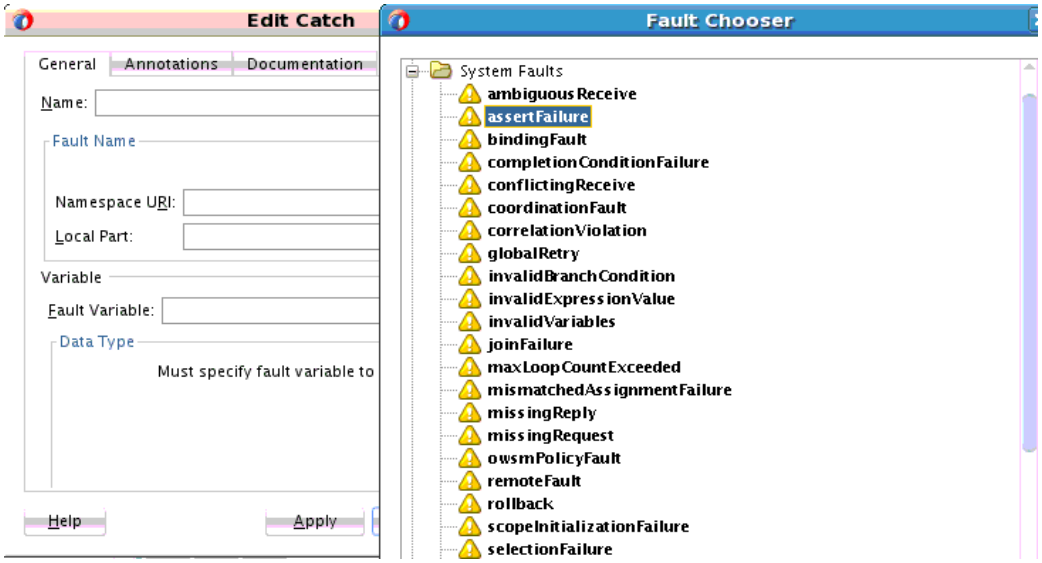
There is no assert activity to trigger this system fault. You can add one to assert an input field.

5. In the **Namespace URI** field, click the **Browse** icon.

   The Fault Chooser dialog is displayed.

6. Select a system fault (for this example, **assertFailure**), and click **OK**. There are many other system faults that can be selected. Figure 12-4 provides details.

Figure 12-4 Fault Chooser Dialog



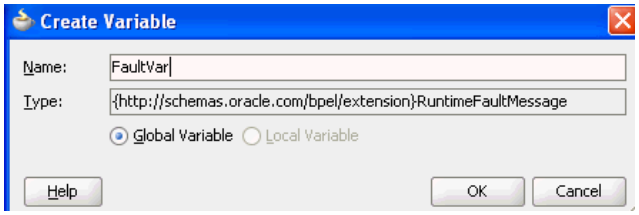
You are returned to the Edit Catch dialog.

7. In the **Fault Variable** field, click the **Create Variable** icon.

   The Create Variable dialog is displayed.

   A name of **FaultVar** and a variable of type **RuntimeFaultMessage** are created. Figure 12-5 provides details.

Figure 12-5 Create Variable Dialog



8. Copy the **RuntimeFault.wsdl** file into the **SOA** > **WSDLs** folder. This is the same location as the BPEL process WSDL file.

9. Click **OK**, and then click **OK** in the Edit Catch dialog.

Create a Fault Response Variable

To create a fault response variable:

1. In the Structure window, right-click the **Variables** folder and select **Create Variable**.

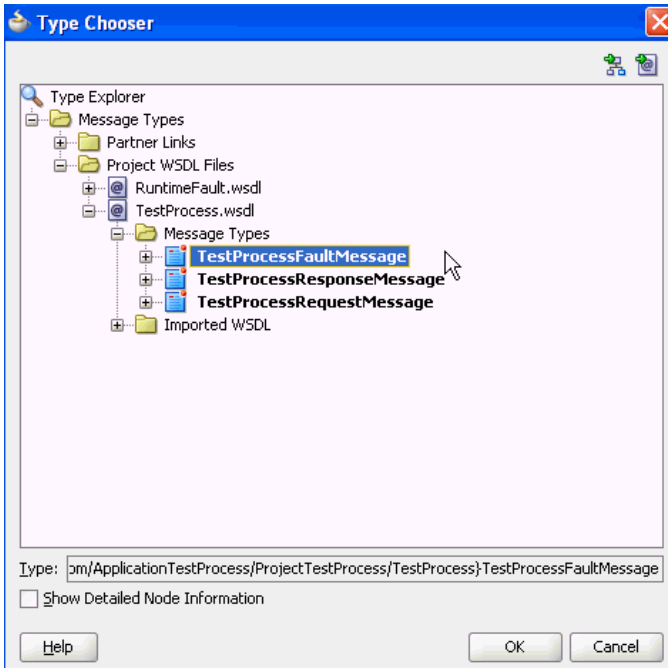
2. In the **Name** field, enter `Faultresponse`.

3. Select **Message Type**.

4. For the **Message Type** field, click the **Browse** icon.

5. Expand **Message Types** > **Project WSDL Files** > **TestProcess.wsdl** > **Message Types** > **TestProcessFaultMessage**, and click **OK**. Figure 12-6 provides details.

Figure 12-6 Type Chooser Dialog



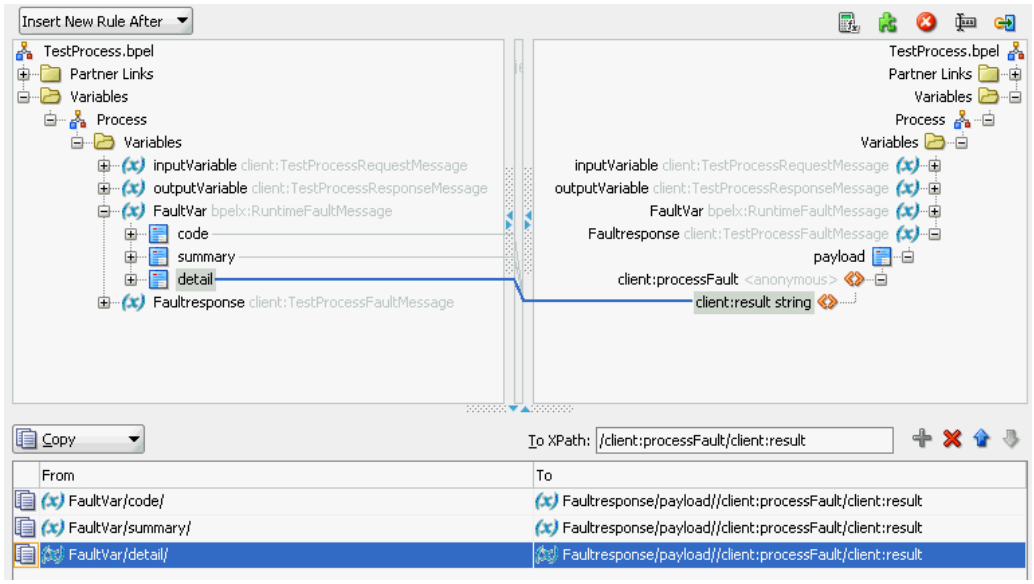
6. In the Create Variable dialog, click **OK**.

Add an Assign Activity to the Catch Activity Branch

To add an assign activity to the catch activity branch:

1. Drag an assign activity into the catch activity block.
2. Double-click the assign activity.
3. Concatenate the **code**, **summary**, and **detail** fields of the **FaultVar** variable to the **FaultResponse** variable, and click **OK**. Figure 12-7 provides details.

Figure 12-7 Edit Assign Dialog



4. In the **Name** field of the **General** tab, enter a name (for this example, `FaultDataForClient`).

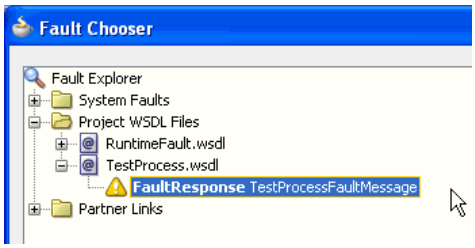
Add a Reply Activity to the Catch Activity Branch

To add a reply activity to the catch activity branch:

1. Drag a **Reply** activity below the **Assign** activity in the catch activity block.
2. Double-click the **Reply** activity.
3. In the **Namespace URI** field, click the **Browse** icon.

   The Fault Chooser dialog is displayed.
4. Expand **Project WSDL Files** > **TestProcess.wsdl**, and select the fault named **FaultResponse**. Figure 12-8 provides details.

Figure 12-8 Fault Chooser Dialog

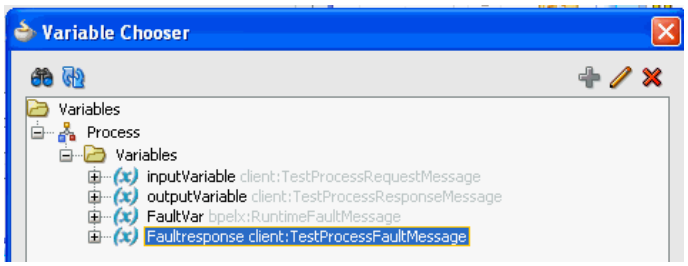


5. In the **Name** field, enter a name (for this example, `ReplyWithFault`).
6. In the **Partner Link** field, click the **Browse** icon.

   The Partner Link Chooser dialog is displayed.
7. Select the same partner link to which the **replyOutput** reply activity is connected, and click **OK**.
8. For the **Variable** field, click the **Browse** icon.

   The Variable Chooser dialog is displayed.
9. Select the **FaultResponse** variable, and click **OK**.

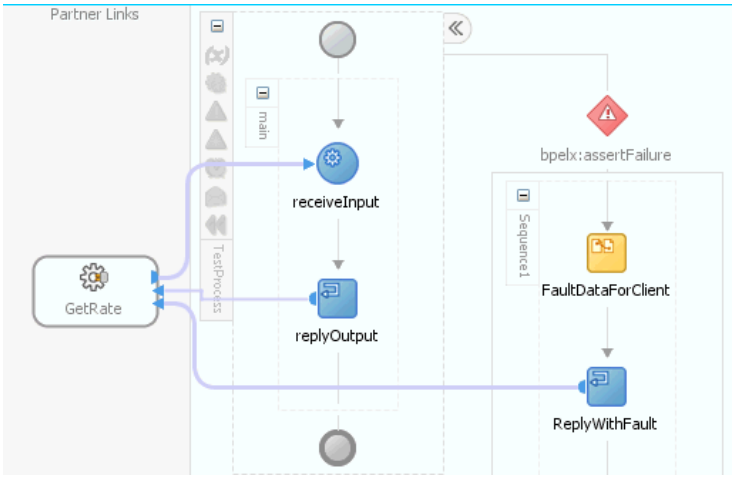
Figure 12-9 Variable Chooser Dialog



10. In the Edit Reply dialog, click **OK**.

    The BPEL process looks as shown in Figure 12-10. Both reply activities are connected to the same partner link.

Figure 12-10 BPEL Process Design



Handling Faults with 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 composite or component. Fault policies are applicable to the faults that result from the invoke activity. Faults can occur because of preassertion, postassertion, invocation, or actual business failures in the target service.

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 a fault policy).
- Email or JMS notify users of errors associated with a condition.
- 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 processes 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. The remote location is in the MDS Repository.

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. The following example provides details about these two properties. In this example, the fault policy files are placed into the SOA part of the Oracle Metadata Services (MDS) Repository shared area.

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

For details about Oracle Mediator fault handling capabilities, see Using Error Handling .

For details about creating a fault policy with Oracle Business Process Management (BPM) Suite, see Chapter "Using Fault Handling in BPM" of *Developing Business Processes with Oracle Business Process Management Studio*.

Understanding How the 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-bindings.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-4`).
- If the fault resolution fails, go to the next possible match (`policy-id-2`).
- 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.

How to Design a Fault Policy for Automated Fault Recovery with the Fault Policy Wizard

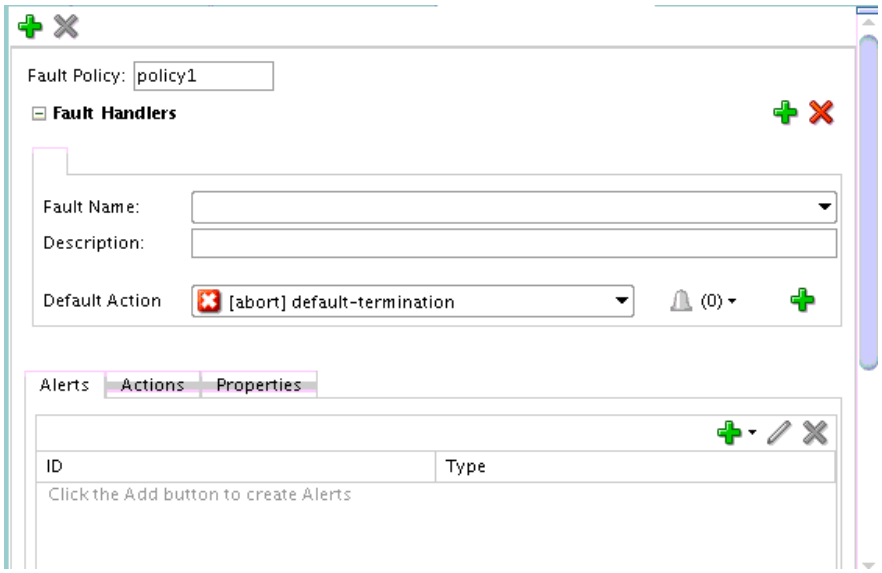
You can design a fault policy with the Fault Policy wizard and associate the fault policy with the fault policy binding file.

To design a fault policy for automated fault recovery with the Fault Policy wizard:

1. From the Oracle JDeveloper main menu, select **File** > **New** > **From Gallery**.
2. In the **Categories** list, select **SOA Tier** > **Faults**.
3. In the **Items** list, select **Fault Policy Document**.

   The Fault Policy Editor is displayed, as shown in Figure 12-11. A single fault policy with a name of **policy1** is initially displayed for configuration.

Figure 12-11 Fault Policy Editor When Initially Displayed



The Fault Policy Editor consists of several sections and tabs. It is recommended that you configure the Fault Policy Editor in the following order:

- **Properties** tab
- **Alerts** tab
- **Actions** tab
- Fault policy name and fault handlers
- Association with the fault policy binding file

Step 1: Defining Property Sets

You first define property sets to associate with JMS alerts, which are defined in Step 2: Defining Alerts. You can associate property sets configuration details such as JMS destinations and connection factories with multiple JMS alerts. For example, for a JMS alert, the destination and queue information and connection factory can be referenced by additional JMS alerts configured in the fault policy.

Note:

You cannot create property sets for email alerts in this release.

---

1. Click the **Properties** tab. Table 12-1 provides details about available fields.

Table 12-1 Property Set Selections

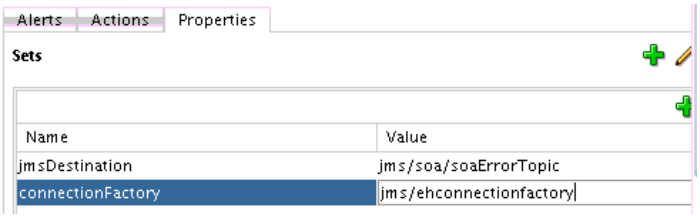
For...	Then...
Email alerts	Email alerts do not support property sets for this release.

Table 12-1 (Cont.) Property Set Selections

For...	Then...
JMS queue alerts	a. Click **Add** to specify the properties and values for JMS alerts. The following properties and associated values are required: • **jmsDestination**: The JNDI name of the configured queue or topic in which the alerts is queued/published. • **connectionFactory**: JNDI name for the configured connection factory to use.

Figure 12-12 shows a property set configured with JMS destination and connection factory values.

Figure 12-12 JMS Property Set Configuration



For an example of a fully-defined fault policy file, including a defined `JMS propertySet` section, see Step 4 of How to Manually Design a Fault Policy for Automated Fault Recovery.

Step 2: Defining Alerts

1. Click the **Alerts** tab. Two types of notification alerts are supported:

   - **Email**: Enables you to configure email recipients to receive alerts when a fault occurs. You must also configure the same email recipients on the **Mailer** tab of the Workflow Notification Properties page in Oracle Enterprise Manager Fusion Middleware Control. For information, see Section "Configuring Human Workflow Notification Properties" of *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

   - **JMS**: Enables you to enqueue the fault to a JMS queue or publish it to a JMS topic. JMS header values can also be specified. The JMS notification can be integrated with a third-party resolution system to handle faults. The third-party resolution system dequeue and subscribes to the targeted queue and topic. Further fine-graining is achieved by consuming messages based on the header property values. The payload type of the JMS message is a text message in XML format. You must also configure JMS queues and topics and connection factories in Oracle WebLogic Server Administration Console. For information, see Section "Configuring Basic JMS System Resources" of *Administering JMS Resources for Oracle WebLogic Server*.

2. Click the **Add** icon. Table 12-2 provides details.

Table 12-2 Alert Selections

If You Select...	Then...
email	You can specify recipients to receive an email alert when a fault occurs. **a.** In the **ID** field, specify an ID or accept the default value. **b.** In the **To** and **CC** fields, specify the email recipients. **Note**: Do *not* select any property sets from the **Property Set** list. The email alert does not support property sets for this release. **c.** When complete, click **OK**.
JMS	You can specify queues to receive a JMS alert when a fault occurs. Two properties are required for configuring a JMS alert. • **jmsDestination**: The JNDI name of the configured queue or topic on which the alert is queued and published. • **connectionFactory**: The JNDI Name for the configured connection factory to use. **a.** In the **ID** field, specify an ID or accept the default value. **b.** In the **Property Set** list, select an existing property set created in Step 1: Defining Property Sets or click **Create Required Properties** to create a new property set with values defined for **jmsDestination** and **connectionFactory**. **c.** In the **Headers** table, optionally specify JMS header values to achieve finer-grained fault consumption for a JMS alert. Both standard and custom external systems can filter their subscriptions based on the configured header properties. **d.** When complete, click **OK**.

Figure 12-13 shows email alert configuration in the Email Properties dialog.

Figure 12-13 Email Alert Configuration

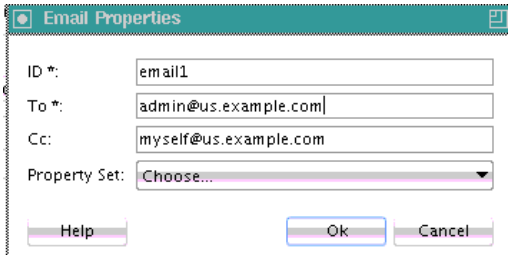
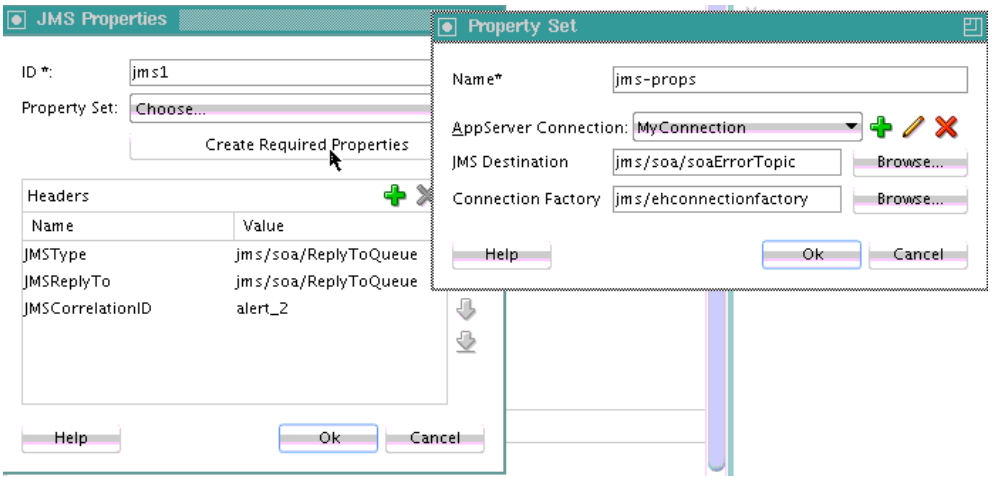


Figure 12-14 shows JMS alert configuration in the JMS Properties dialog. For this example, both property sets (defined by clicking **Create Required Properties** to invoke the Property Set dialog) and headers are defined.

Figure 12-14 JMS Alert Configuration



For an example of a fully-defined fault policy file, including a defined `Alerts` section, see Step 4 of How to Manually Design a Fault Policy for Automated Fault Recovery.

Step 3: Defining Actions

1. Click the **Actions** tab. By default, all types of actions are automatically selected. Figure 12-15 provides details.

Figure 12-15 Actions Section of Fault Policy Editor

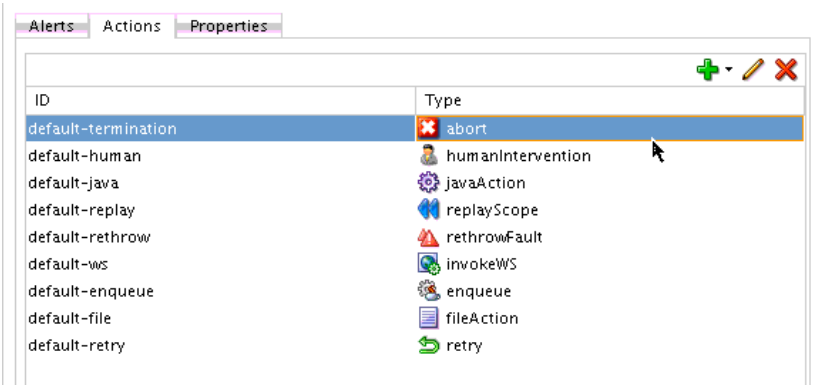


Table 12-3 describes the available action types.

Table 12-3 Supported Action Types

Action	Description
Abort	Terminates the entire business flow.
Human intervention	Causes the current activity to stop processing. Human intervention from Oracle Enterprise Manager Fusion Middleware Control is required to handle the fault. For information, see "Recovering from Faults in a Business Flow Instance" of *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

Table 12-3 (Cont.) Supported Action Types

Action	Description
Java action:	Enables you to execute an external Java class. For more information, see How to Use a Java Action Fault Policy.
Replay scope	Raises a replay fault.
Rethrow fault	Sends the fault to the BPEL fault handlers (catch activities in scope activities). If none are available, the fault is sent up.
Enqueue	Enqueues a rejected message to a JMS queue as a JMS message with the appropriate context and payload. For additional configuration information, see Section "JMS Queue" in *Understanding Technology Adapters*.
Invoke WS:	Handles a rejected message by calling a web service. For additional configuration information, see Section "Web Service Handler" in *Understanding Technology Adapters*.
File action	Creates an error handler for messages by storing a rejected message in a file. For additional configuration information, see Section "File" in *Understanding Technology Adapters*.
Retry	Provides the following options 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. For more information about retries, see Table 12-6.

For an example of a fault policy file with a defined `Actions` section, see Step 4 of How to Manually Design a Fault Policy for Automated Fault Recovery.

Step 4: Defining Fault Names and Policies

1. Define the fault name, description, and default action of the fault policy in the upper section of the Fault Policy Editor. Table 12-4 provides details.

Table 12-4 Fault Policy Editor - Upper Section

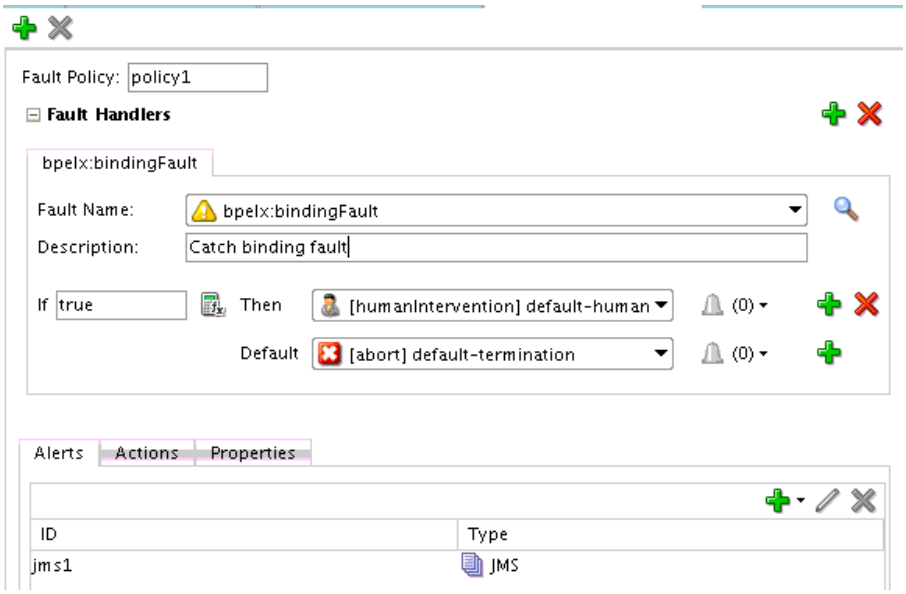
Element	Description
Add Fault Policy icon (upper left corner)	You can also add additional fault policies for configuration to a single policy document. Click the **Add** icon in the upper left corner to add an additional fault policy. All polices are then displayed in the column on the far left of the Fault Policy Editor. You can click the policy that you want to define.
Delete Fault Policy	Delete a selected fault policy.
Fault Policy	Enter a name for the fault policy or accept the default name of `policy`*`number`*.

Table 12-4 (Cont.) Fault Policy Editor - Upper Section

Element	Description
Add Fault icon (upper right corner)	Click to add a fault.
Delete Fault	Click to delete a fault.
Fault Name	Select a standard type of fault to catch. This list shows the system faults (binding, Oracle Mediator, or remote) or service (business) fault that you can select.
Description	Enter an optional description. The description is persisted into the audit trail during runtime.
Default Action	Perform the following tasks in this section: **a.** From the list, select the default action to perform when this fault occurs (for example, abort, rethrow, retry, and so on). The actions available for selection are based on the actions you retained or deleted in Step 3: Defining Actions. or **a.** Click the **Add** icon to add an if-then condition to the fault policy. This selection displays the **If**, **Then**, and **Default** fields. For example, if you specify a condition in the **If** field (the default is **true**), you can select an action (for example, human intervention) to be invoked in the **Then** field. If the condition is not true, you can select the default action to occur (for example, abort) in the **Default** field. **b.** In the **If** field, enter a condition or click the **Expression Builder** icon to build an XPath expression condition. **c.** In the **Then** field, specify the condition to invoke if the condition in the **If** field evaluates to true. **d.** In the **Default** field, specify the condition to invoke if the condition in the **If** field evaluates to false. **e.** Click the **Alert** icon to the left of the **Add** icon to select the type of alert to send when this condition occurs. The alert types available for selection are displayed in the **Alerts** tab in this dialog. You can specify multiple alerts on a condition.

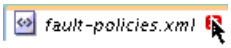
When complete, the Fault Policy Editor looks as shown in Figure 12-16.

Figure 12-16 Fault Policy Editor With Fault Name, Description, and Default Actions Defined



2. Above the SOA Composite Editor, close the fault policy file, and click **Yes** when prompted to save your changes. Figure 12-17 provides details.

Figure 12-17 Save Fault Policy Changes



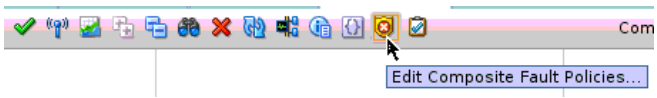
Policy configuration is now complete. You are now ready to associate the fault policy with the fault policy bindings.

Step 5: Defining the Fault Policy Bindings for the Fault Policy

After creating a fault policy with the Fault Policy wizard, you associate the fault policy with a fault policy bindings file. The fault policy bindings file associates the policies defined in the fault policy file with service components, service binding components, or reference binding components in the SOA composite application.

1. Open the SOA Composite Editor.
2. Click the icon above the SOA Composite Editor to define the fault policy bindings for this fault policy. Figure 12-18 provides details.

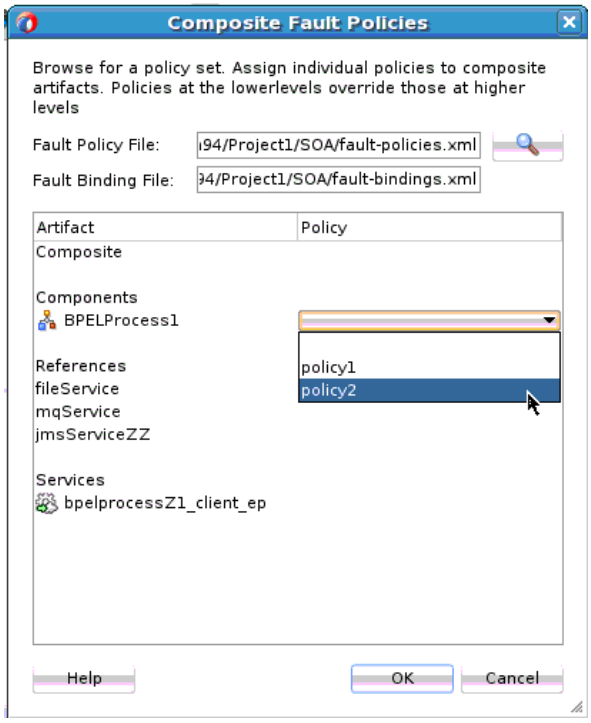
Figure 12-18 Fault Policy Binding Icon



The Composite Fault Policies dialog is displayed.

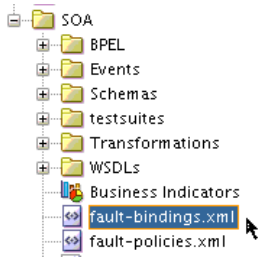
3. If you want to specify a different fault policy file (for example, one created in the file directory or MDS Repository), click the **Browse** icon to the right of the **Fault Policy File** field.
4. In the **Policy** column for the SOA composite application, service binding component, or reference binding component, select the fault policy to attach. Figure 12-19 provides details.

Figure 12-19 Composite Fault Policies Dialog



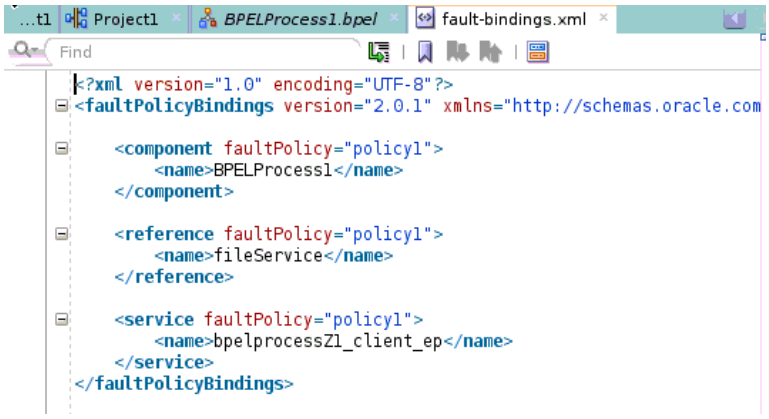
5. In the **SOA** folder in the Applications window, select the `fault-bindings.xml` file to view its contents. Figure 12-20 provides details.

Figure 12-20 Selection of fault-bindings.xml File



The file looks as shown in Figure 12-21.

Figure 12-21 fault-bindings.xml file



How to Manually Design a Fault Policy for Automated Fault Recovery

This section describes how to manually design a fault policy. The recommended approach is to design a fault policy with the Fault Policy wizard, as described in How to Design a Fault Policy for Automated Fault Recovery with the Fault Policy Wizard.

Manually Creating a Fault Policy File for Automated Fault Recovery

To manually create 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.

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

   If the fault policy file is located in a file system, use the following format.

```
<property
name="oracle.composite.faultPolicyFile">file:/project/apps/fault-policies.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.
     - You can associate a single or multiple alerts with a condition to be delivered (by email, JMS queue, or log file) when a specific error condition occurs.

   Table 12-5 provides examples of the `condition` section in the fault policy file. All actions defined in the `condition` section must be associated with an action in the `action` section.

Table 12-5 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>`

Two user notification alerts are defined for the condition. Select the type of user notification alert to create when a fault occurs (for example, an email alert, a JMS queue alert, or a log file alert).	`<condition>`
	`  <alert ref = "ora-jms"/>`
	`  <alert ref = "ora-email"/>`
	`  <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. Validation of fault policy files is done during deployment. If you change the fault policy, you must redeploy the SOA composite application that includes the fault policy.

   Table 12-6 provides several examples of the `action` section in the fault policy file. You can provide automated recovery actions for some faults. In all recovery actions except retry and human intervention, the framework performs the actions synchronously.

Table 12-6 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 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.
	`<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 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>`

Table 12-6 (Cont.) Use of action Section in the Fault Policy File

Recovery Actions	Fault Policy File Syntax
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.

A fault policy file with fully-defined `condition`, `action`, and `alert` sections looks as follows:

Note:

- Fault policy file names are not restricted to one specific name. However, they must conform to the `fault-policy.xsd` schema file.
- This fault policy file provides an example of catching faults based on fault names. You can also catch faults based on message types, or on both:

  ```
  <faultName name="myfault" type="fault:faultType">
  ```

```
<?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="2.0.1" id="ModifyAndRecover"
               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>
      <!-- Handle remoteFault system exceptions -->
      <faultName xmlns:bpelx="http://schemas.oracle.com/bpel/extension"
 name="bpelx:remoteFault">
        <condition>
        <!--<test>$fault.code="1"</test>-->
          <alert ref = "ora-jms"/>
          <alert ref = "ora-email"/>
       <action ref="default-human-intervention"/>
        </condition>
      </faultName>
           <faultName xmlns:bpelx="http://schemas.oracle.com/bpel/extension"
 name="bpelx:bindingFault">
        <condition>
          <action ref="default-human-intervention"/>
```

```
        </condition>
      </faultName>          </Conditions>
       <Alerts>
          <Alert id="ora-email">
           <email>
             <To>joe.smith@example.com</To>
                 <CC>joe.smith@example.com</CC>
           </email>
          </Alert>
          <Alert id="ora-jms">
        <JMS propertySet="jms-props">
          <Headers>
            <property name="correlationId">myvalue</property>
             <property name="correlationId1">myvalue1</property>
          </Headers>
        </JMS>
      </Alert>
    </Alerts>
    <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="ora-retry-with-human-intervention">
        <retry>
          <retryCount>1</retryCount>
          <retryInterval>2</retryInterval>
          <exponentialBackoff/>
          <retryFailureAction ref="default-terminate"/>
        </retry>
      </Action>
      </Actions>
      <Properties>
       <propertySet name="jms-props">
         <property name="jmsDestination">MyQueue</property>
         <property
 name="connectionFactory">jms/fabric/ehconnectionfactory</property>
     </propertySet>
     </Properties>
  </faultPolicy>
</faultPolicies>
```

Associating a Fault Policy with Fault Policy Binding

Note:

The fault policy binding file must be named `fault-bindings.xml`. This conforms to the `fault-bindings.xsd` schema file.

To associate a fault policy with fault policy binding:

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 Manually Creating a Fault Policy File for Automated Fault Recovery.

   This fault policy bindings file associates the fault policies defined in the `fault-policies.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>
```

Additional Fault Policy and Fault Policy Binding File Samples

This section provides additional samples of fault policy and fault policy binding files. The following example shows the `fault-policies.xml` file contents.

```
<?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>
                  <alert ref = "ora-email"/>
                <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>
```

The following example shows the `fault-bindings.xml` file that associates the fault policies defined in `fault-policies.xml`.

```
<?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>
```

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 the following example. In this case, only the first rejection handler defined (for this example, `ora-queue`) is executed.

```
<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>
```

How to Execute a Fault Policy

You deploy a fault policy as part of a SOA composite application. After deployment, you can perform the fault recovery actions from Oracle Enterprise Manager Fusion Middleware Control. Actions such as terminate, retry, rethrow, and Java are retried as part of composite execution. No explicit user execution is required. The human intervention action can be manually executed in 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 *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

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 the 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 the following example.

```
<Action id="ora-java">
  <javaAction className="mypackage.myclass"
    defaultAction="ora-human-intervention" propertySet="prop-for-billing">
   <!--defaultAction is a required attribute, but propertySet is optional-->
   <!-- attribute-->
     <ReturnValue value="RETRY" ref="ora-retry"/>
     <!--value is not nilable attribute & cannot be empty-->
     <ReturnValue value="RETRHOW" ref="ora-rethrow-fault"/>
  </javaAction>
</Action>
```

Table 12-7 provides an example of `ReturnValue` use.

Table 12-7 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`.
`<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 value 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 the following example:

```
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`.
- The fault policy class is packaged and deployed in either of two ways:
  - Package the Java class with the SOA composite application.
  - If the Java class must be shared by multiple SOA composite applications, place it in the shared location (for example, `$MW_HOME/soa/soa/ modules/`

`oracle.soa.ext_11.1.1`). The shared location includes a `readme` file that describes how to place the Java class to make it available in the class path.

The following example shows the 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). The following example provides details:

```
public class BPELFaultRecoveryContextImpl extends BPELXExecLetUtil implements
IBPELFaultRecoveryContext,  IFaultRecoveryContext{
...
}
```

Oracle BPEL Process Manager-specific data is available with `IBPELFaultRecoveryContext`, as shown in the following example:

```
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;
}
```

The following example provides an example of `javaAction` implementation.

```
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());
         ...
        }
```

How to Design Fault Policies for Oracle BPM Suite

You can design and execute fault policies for Oracle BPM Suite. For more information, see Chapter "Using Fault Handling in BPM" of *Developing Business Processes with Oracle Business Process Management Studio*.

What You May Need to Know About Designing a Fault Policy in a Synchronous BPEL Process

When designing a fault policy in a synchronous process, do not specify the following actions. These actions cause dehydration in a synchronous process and can lead to timeouts.

- Retry
- Human intervention
- Terminate

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 the following code after `ora-retry`:

```
<Action id="ora-retry">
       <retry>
          <retryCount>2</retryCount>
          <retryInterval>2</retryInterval>
          <exponentialBackoff/>
       </retry>
  </Action>
```

The following actions are performed:

- The invoke activity is attempted (using the above-mentioned fault policy code to handle the fault).
- Two retries are attempted at increasing intervals (after two seconds, then after four seconds).
- If all retry attempts fail, the following actions are performed:
  - A detailed fault error message is logged in the audit trail.
  - The instance is marked as `open.faulted` (in-flight state).

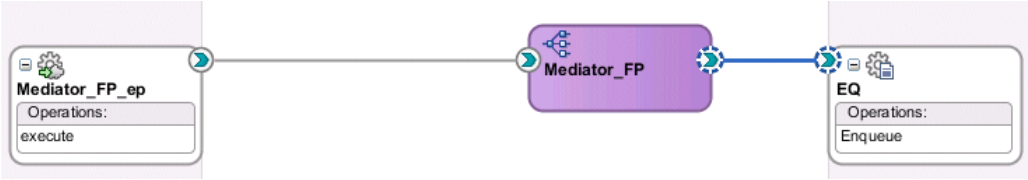
  - The instance is picked up and the invoke activity is re-attempted.

- Recovery may also fail. In that case, the invoke activity is re-executed. Additional audit messages are logged.

What You May Need to Know About Binding Level Retry Execution Within Fault Policy Retries

If you are testing retry actions on adapters with both JCA-level retries for the outbound direction and a retry action in the fault policy file for outbound failures, the JCA-level (or binding level) retries are executed within the fault policy retries. For example, assume you have designed the application shown in Figure 12-22:

Figure 12-22 SOA Composite Application



You specify the retry parameters, as shown below, in the `composite.xml` file:

```
<property name="jca.retry.count" type="xs:int" many="false"
  override="may">2</property>
<property name="jca.retry.interval" type="xs:int" many="false"
  override="may">2</property>
<property name="jca.retry.backoff" type="xs:int" many="false"
  override="may">2</property>
```

In the fault policy file for the **EQ** reference binding component for the outbound direction, you specify the actions shown in the following code:

```
<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)

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.

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 Repository 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">
```

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.

How to Get Fault Details with the getFaultAsString XPath Extension Function

The following example shows how to use this function.

```
<catchAll>
   <sequence>
      <assign>
         <from expression="bpelx:getFaultAsString()"/>
         <to variable="faultVar" part="message"/>
      </assign>
      <reply faultName="ns1:myFault" variable="faultVar" .../>
   </sequence>
</catchAll>
```

For more information, see getFaultAsString.

Throwing Internal Faults with the Throw Activity

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"/>
```

How to Create a Throw Activity

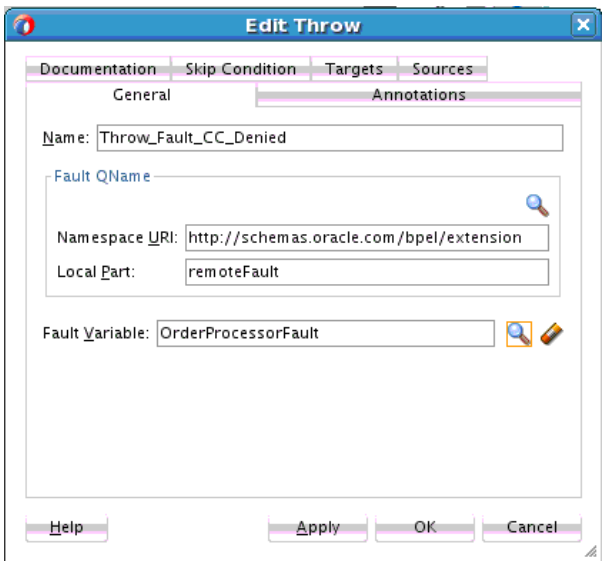
To create a throw activity:

1. In the Components window, 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 12-23 provides an example of a completed Throw dialog.

Figure 12-23 Throw Dialog



7. Click **Apply**, then **OK**.

What Happens When You Create a Throw Activity

The following code shows the throw activity in the `.bpel` file after design completion. The `OrderProcessor` process terminates after executing this throw activity.

```
<throw name="Throw_Fault_CC_Denied"
    faultName="client:OrderProcessorFault"/>
```

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:

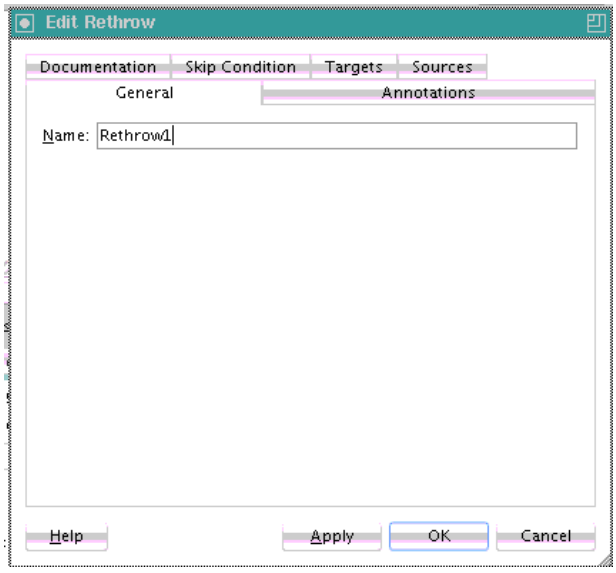
This activity is supported in BPEL version 2.0 projects.

How to Create a Rethrow Activity

To create a rethrow activity:

1. In the Components window, 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 12-24.

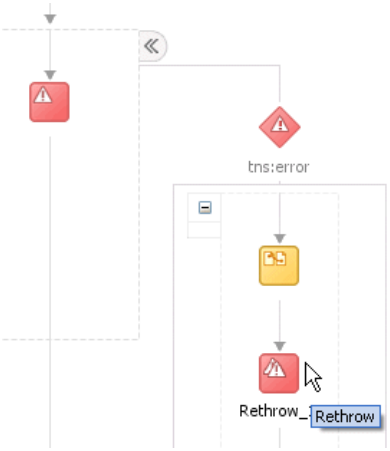
Figure 12-24 Rethrow Dialog



5. Click **Apply**, then **OK**.

When complete, design can look as shown in Figure 12-25.

Figure 12-25 Throw Activity in BPEL Process



What Happens When You Rethrow Faults

The following example shows the `.bpel` file after design is complete for a rethrow activity. The rethrow activity is inside a fault handler (catch 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>
```

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.

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 the following example:

```
<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:

- Invoking a Synchronous Web Service from a BPEL Process for synchronous interactions
- Using Conditional Branching in a BPEL Process for setting up the conditional structure

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 Invoking an Asynchronous Web Service from a BPEL Process.

Managing a Group of Activities with a Scope Activity

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.

The following example shows a scope named `Scope_FulfillOrder`. This scope invokes the `FulfillOrder` Oracle Mediator component, which determines the shipping method for the order.

```
<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>
```

How to Create a Scope Activity

To create a scope activity:

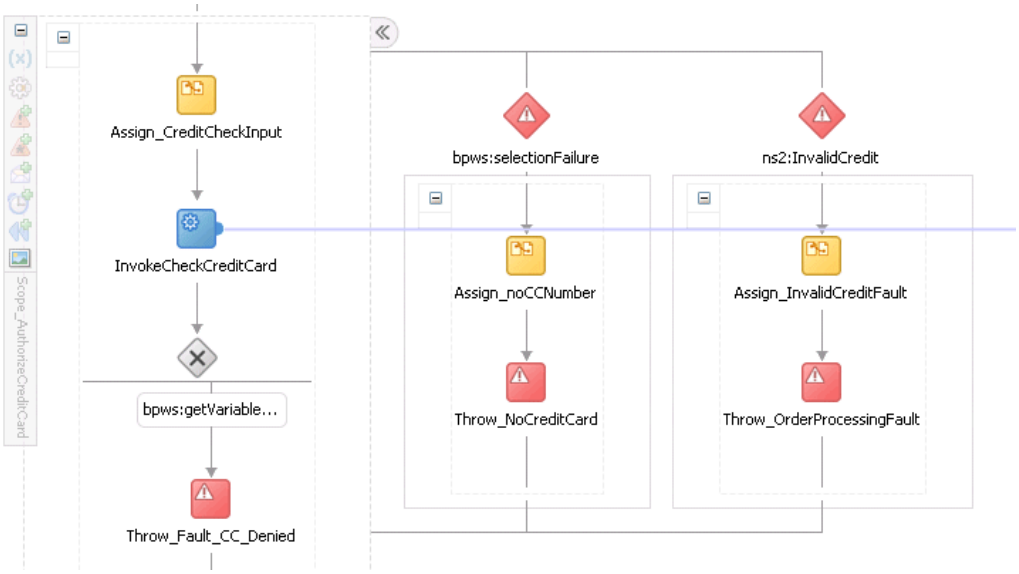
1. In the Components window, expand **BPEL Constructs**.
2. Drag a **Scope** activity into the designer.
3. Open the **Scope** activity by double-clicking it or by single-clicking the **Expand** icon.
4. From the Components window, drag and define activities to build the functionality within the scope. Figure 12-26 provides details.

Figure 12-26 Expanded Scope Activity

5. Click **OK**.

When complete, scope activity design can look as shown in Figure 12-27. This example shows a **Scope_AuthorizeCreditCard** scope activity.

Figure 12-27 Scope Activity After Design Completion



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.

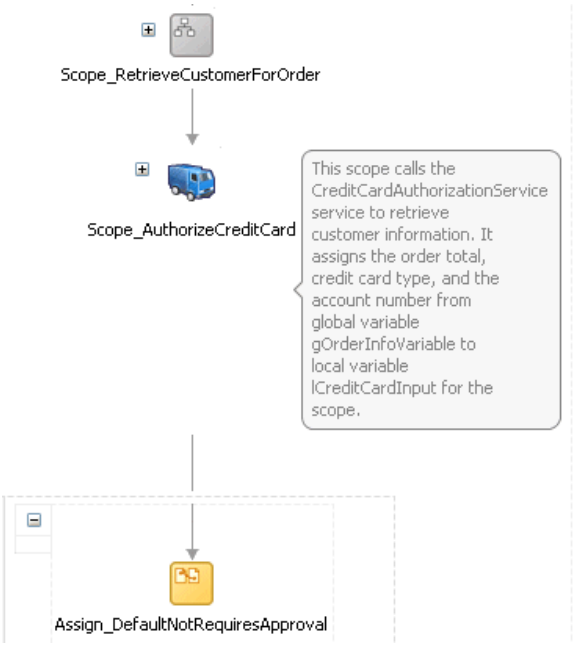
To add descriptive notes and images to a scope activity:

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 12-28.

Figure 12-28 Scope with Descriptive Note and Modified Image



5. To edit the note, double-click it.

What Happens After You Create a Scope Activity

The following example 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.

```
<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>
```

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.

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.

The following example 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.

```
<faulthandlers>
   <catch faultName="x:foo">
         <empty/>
      </catch>
   <catch faultVariable="bar">
         <empty/>
      </catch>
   <catch faultName="x:foo" faultVariable="bar">
         <empty/>
      </catch>
   <catchAll>
         <empty/>
      </catchAll>
</faulthandlers>
```

What You May Need to Know About the idempotent Property and Fault Handling

If the `idempotent` deployment descriptor property is set to `false` in the `composite.xml` file and the invocation of a partner link fails, recovery does not start from the invoke activity. Relying on the `idempotent` property for retrying the invoke activity is not recommended. Instead, recovery is attempted by fault handling you have designed into the BPEL process (such as with a catchAll activity). As a best practice, Oracle recommends that you instead use a fault policy to retry the invoke activity.

Table 12-8 describes the behavior when the `idempotent` property is set to `false` and partner link invocation either succeeds or fails.

Table 12-8 Recovery Behavior When the idempotent Property Is Set to False

If Partner Link Invocation Is...	Then...
Successful	The invoke activity is dehydrated immediately after execution and recorded in the dehydration store.
Unsuccessful, and your BPEL process includes fault handling, such as a catchAll activity	Recovery is started from the catchAll activity and *not* from the invoke activity.
Unsuccessful, and your BPEL process includes a fault policy	The fault policy is used to attempt recovery of the invoke activity. This is the recommended approach.

For example, assume your BPEL process includes the following design:

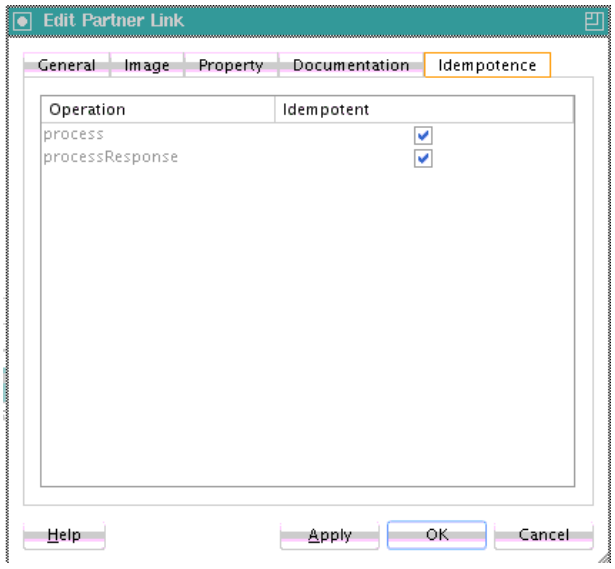
- An invoke activity invokes a partner link (for this example, named `myPartnerLink`).
- The `idempotent` deployment descriptor property is set to `false` in the `composite.xml` file.

  ```
  <property name="bpel.partnerLink.myPartnerLink.idempotent">false</property>
  ```

This setting causes the BPEL process to dehydrate immediately after execution of this activity and be recorded in the dehydration store.

You can also set this property to `false` in the Edit Partner Link dialog. Figure 12-29 provides details.

Figure 12-29 Idempotence Tab of Edit Partner Link Dialog



For more information, see Managing Idempotence at the Partner Link Operation Level.

- A catchAll activity error handler in a scope activity catches faults and throws a rollback fault.

If the invocation by the invoke activity to the partner link fails, recovery starts from the catchAll activity error handler, and *not* from the invoke activity. The recovery from the catchAll activity can be observed in the flow activity for the BPEL process in Oracle Enterprise Manager Fusion Middleware Control.

This is by design. The `idempotent` property setting is checked *after* execution of the invoke activity. If the execution failed and an exception is raised, the `idempotent` property setting is never reached. The BPEL process service engine saves the instance right after opening the catchAll activity. The instance must be saved because the `idempotent` property is set to `false`. This is why recovery resumes in the catchAll activity.

Oracle recommends that you instead recover the failed invoke activity with a fault policy. For more information about creating fault polices, see Handling Faults with the Fault Management Framework.

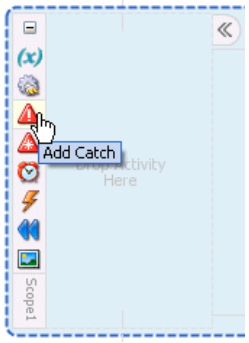
For more information about the `idempotent` property, see Introduction to Deployment Descriptor Properties.

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 12-30 provides details.

Figure 12-30 Add Catch



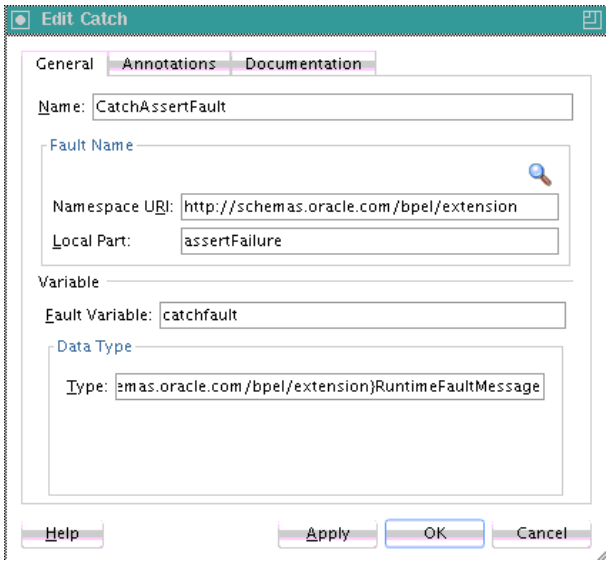
This creates a catch activity on the right side of the scope activity.

2. Double-click the **Catch** activity.
3. Optionally enter a name.
4. To the right of the **Namespace URI** field, click the **Search** icon to select the fault.
5. Select the fault in the Fault Chooser dialog, and click **OK**.

   The namespace URI for the selected fault displays in the **Namespace URI** field. Your fault selection also automatically displays in the **Local Part** field.

   Figure 12-31 provides an example of a Catch dialog. This example shows the **selectionFailure** catch activity of a **Scope_AuthorizeCreditCard** scope activity. This catch activity catches orders in which the credit card number is not provided.

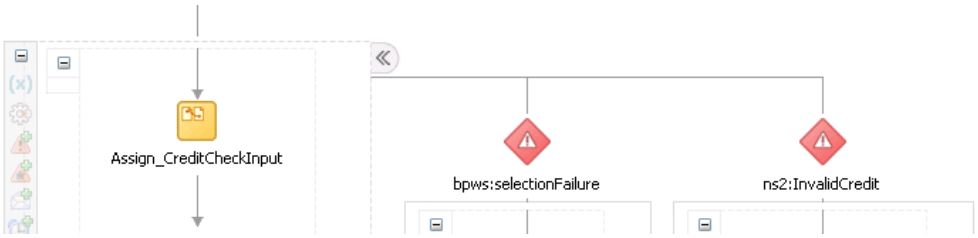
Figure 12-31 Catch Dialog



6. Design additional fault handling functionality.
7. Click **OK**.

   Figure 12-32 provides an example of two catch activities for the **Scope_AuthorizeCreditCard** scope activity. The second catch activity catches credit types that are not valid.

Figure 12-32 Catch Activities in the Designer



What Happens When You Create a Catch Activity in a Scope

The following example 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.

```
<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>
```

If no catch or catchAll activity 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 Stopping a Business Process Instance with the Terminate Activity in BPEL 1.1) had been performed.

How to Insert No-Op Instructions into a Business Process with an Empty Activity

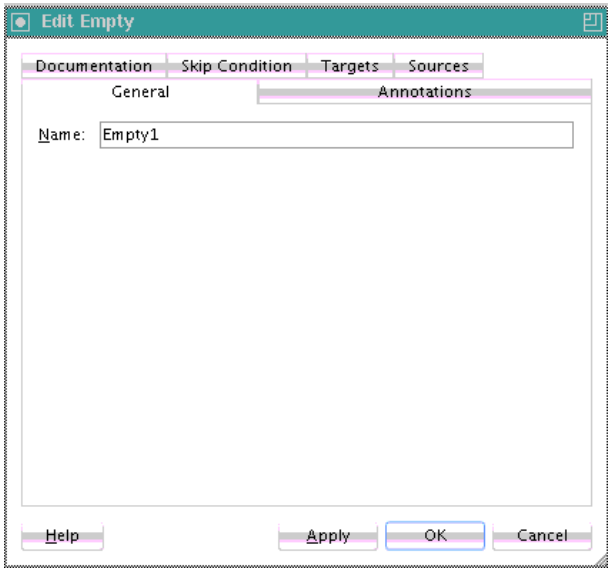
There is often a need to use an activity that does nothing. An example is when a fault must be caught and suppressed. In this case, you can use the empty activity to insert a no-op instruction into a business process.

To create an empty activity:

1. In the Components window, 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 12-33.

Figure 12-33 Empty Activity



4. Optionally enter a name.
5. Click **OK**.

What Happens When You Create an Empty Activity

The syntax for an empty activity is shown in the following example:

```
<empty standard-attributes>
   standard-elements
 </empty>
```

Re-executing Activities in a Scope Activity with the Replay Activity

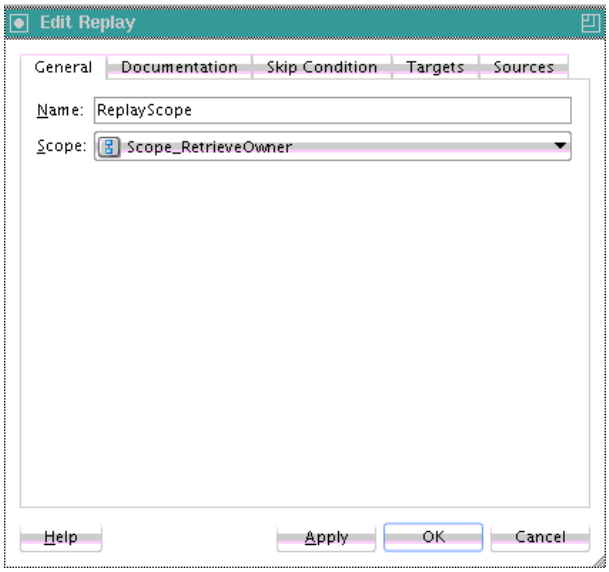
You can create a replay activity inside a scope activity to re-execute all of the activities inside the scope.

How to Create a Replay Activity

To create a replay activity:

1. In the Components window, 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 12-34.

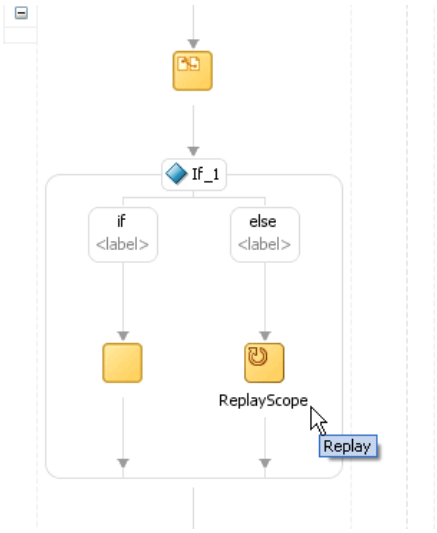
Figure 12-34 Replay Dialog



6. Click **Apply**, then click **OK**.
7. Continue with the design of your scope activity.

   When complete, design of the scope activity can look as shown in Figure 12-35.

Figure 12-35 Replay Activity in a Scope Activity



What Happens When You Create a Replay Activity

The following example 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.

```
<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="Scope_RetrieveOrder"/>
               </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"/>
```

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.

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 the . 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 to perform compensation.
- In the compensation handler of the scope that immediately encloses the scope for which to perform compensation.

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.

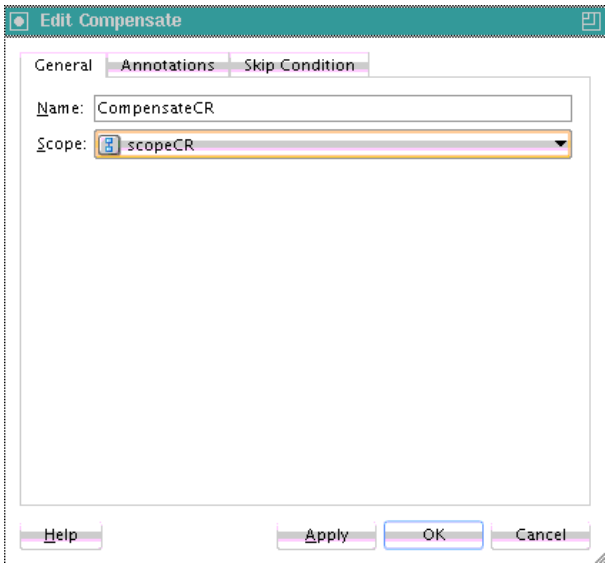
The compensate form, in which the scope name is omitted in a compensate activity, explicitly invokes this default behavior. This is useful when an enclosing fault or compensation handler must perform additional work, such as updating variables or sending external notifications, in addition to performing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

How to Create a Compensate Activity

To create a compensate activity:

1. In the Components window, 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 12-36.

Figure 12-36 Compensate Activity



5. Click **Apply**, then **OK**.

What Happens When You Create a Compensate Activity

If a scope 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 the following example:

```
<compensate scope="ncname"? standard-attributes>
    standard-elements
  </compensate>
```

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. Use this activity only 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.

How to Create a compensateScope Activity

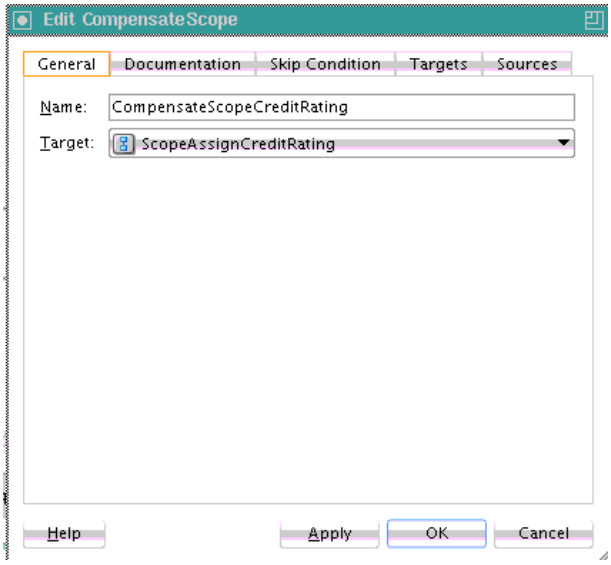
Note:

This activity is supported in BPEL 2.0 projects.

To create a compensateScope activity:

1. In the Components window, expand **BPEL Constructs**.
2. Drag a **CompensateScope** activity into the designer.
3. Double-click the **CompensateScope** activity.
4. In the **Target** list, select a specific scope activity in which to invoke the compensation handler. Figure 12-37 provides details.

Figure 12-37 CompensateScope Activity



5. Click **Apply**, then **OK**.

What Happens When You Create a compensateScope Activity

The following example 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.

```
<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>
```

Stopping a Business Process Instance with a Terminate or Exit Activity

You can stop a business process instance with either of the following activities:

- Exit activity (in a BPEL version 2.0 project)
- Terminate activity (in a BPEL version 1.1 project)

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:

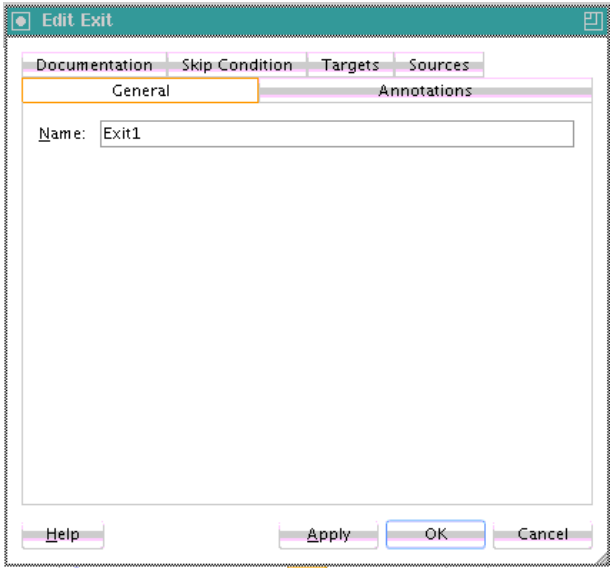
Any open conversations are also impacted by the exit activity. For example, other partners interacting with the process may wait for a response that never arrives.

How to Create an Exit Activity

To create an exit activity:

1. In the Components window, 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 12-38.

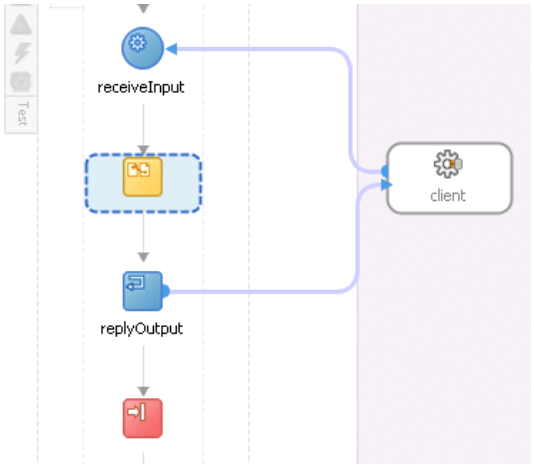
Figure 12-38 Exit Activity



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 12-39.

Figure 12-39 Exit Activity in a BPEL Process



What Happens When You Create an Exit Activity

The following example shows the `.bpel` file after design is complete for an exit activity.

```
<sequence>
    <!-- receive input from requester -->
    <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 requester -->
    <reply name="replyOutput" partnerLink="client"
       portType="tns:Test" operation="process" variable="output"/>
    <exit/>
  </sequence>
```

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.

How to Create a Terminate Activity

To create a terminate activity:

1. In the Components window in Oracle JDeveloper, expand **BPEL Constructs**.
2. Drag a **Terminate** activity into the designer. Figure 12-40 provides an example.

Figure 12-40 Terminate Activity

3. Double-click the **terminate** activity.
4. Optionally enter a name.
5. Click **OK**.

What Happens When You Create a Terminate Activity

The syntax for the `terminate` activity is shown in the following example. This stops the business process instance.

```
<terminate standard-attributes>
    standard-elements
</terminate>
```

Throwing Faults with Assertion Conditions

You can specify an assertion condition in BPEL versions 1.1 and 2.0 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.

You can select when to execute a condition:

- Preassert: This condition is executed before the invoke or reply activity send out the outbound message.
- Postassert: This condition is executed after an invoke activity, receive activity, or onMessage branch receives the inbound message.

How to Create Assertion Conditions

You can create assertion conditions in the following activities:

- In message exchange activities such as invoke activities, receive activities, reply activities, and OnMessage branches
- In standalone assert activities for specifying XPath expressions

To create assertion conditions in invoke activities, receive activities, reply activities, and OnMessage branches:

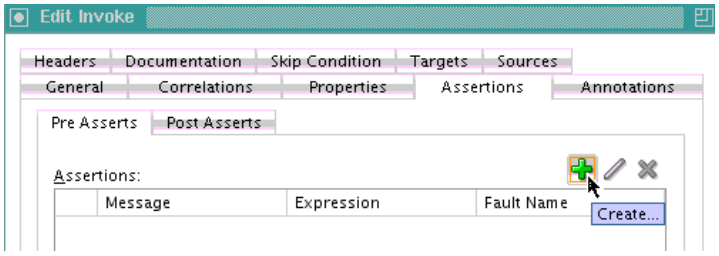
1. In the , double-click the BPEL process service component.
2. In the Components window, expand **BPEL Constructs**.
3. Drag a **Receive** activity, **Invoke** activity, **Pick** activity, or **Scope** activity into the designer.
4. Expand the **Receive**, **Invoke**, or **onMessage** branch of the **Pick** or **Scope** activity.
5. Click the **Assertions** tab.
6. If you are creating an assertion for a BPEL 2.0 project, perform the following tasks. Otherwise, go to Step 6.
   a. Select when to execute the condition. Table 12-9 provides details.

Table 12-9 Assertion Condition Tabs

To Create A...	Select The...
Preassertion condition	**Pre Asserts** tab
Postassertion condition	**Post Asserts** tab

   b. Click the **Add** icon, as shown in Figure 12-41.

Figure 12-41 Add Icon of Assertions Tab in BPEL 2.0

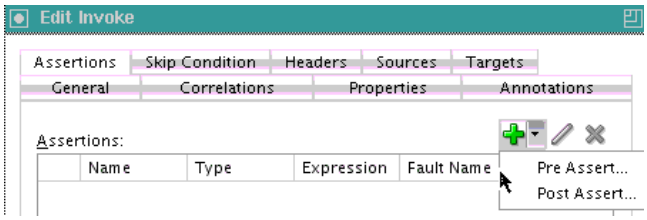


The Assert dialog is displayed.

7. If you are creating an assertion for a BPEL 1.1 project, perform the following tasks.

   a. Click the **Add** icon, as shown in Figure 12-42.

   Figure 12-42 Add Icon of Assertions Tab in BPEL 1.1

   b. Select when to execute the condition. Table 12-10 provides details.

   Table 12-10 Condition Execution Options

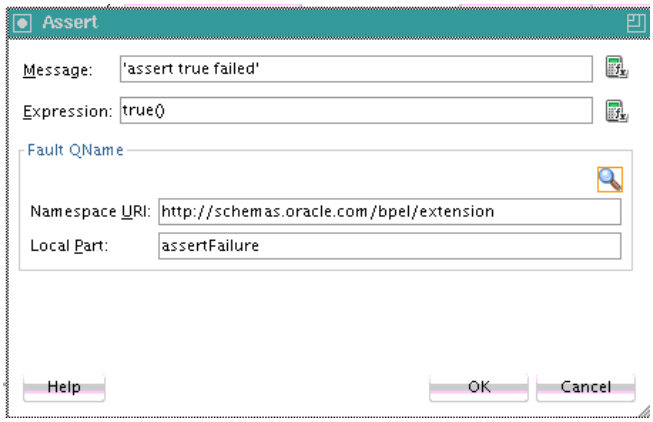
Element	Description
Pre Assert	If selected, the condition is executed before the invoke or reply activity send out the outbound message. **Note**: A fault policy does not handle faults thrown from a preassert condition. Only faults thrown from a postassert condition are supported. For more information about fault policies, see Handling Faults with the Fault Management Framework.
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 12-43. For this example, **Post Assert** was selected for an assertion condition on a receive activity in a BPEL 2.0 project.

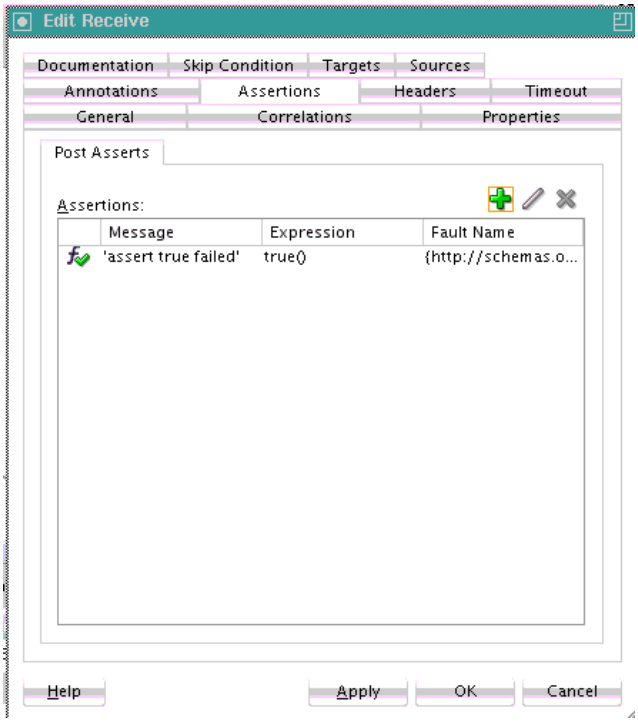
   a. Select the **Fault QName** to be thrown by clicking the **Search** icon and selecting an existing fault from the Fault Chooser dialog. You can also provide your own values for the **Namespace URI** and **Local Part** fields of the fault. If you do not specify anything for the **Fault QName**, then a `bpelx:assertFailure` fault is thrown.

   Figure 12-43 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 12-44.

Figure 12-44 Assertions Tab with Data

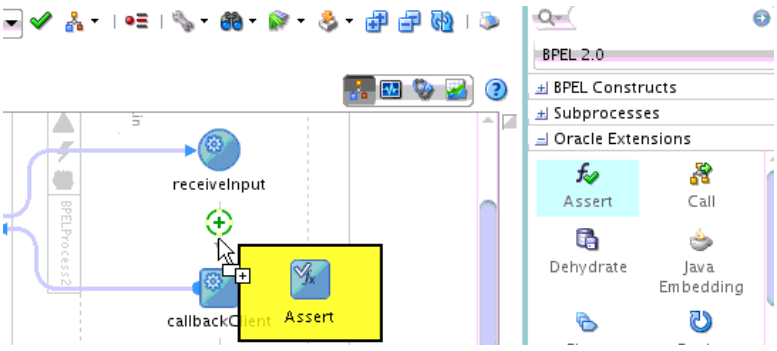


10. Click **Apply**, then **OK**.

To create an assertion condition in standalone assert activities:

1. In the , double-click the BPEL process service component.
2. In the Components window, expand **Oracle Extensions**.
3. Drag an **Assert** activity into the designer, as shown in Figure 12-45.

Figure 12-45 Assert Activity in Components Window

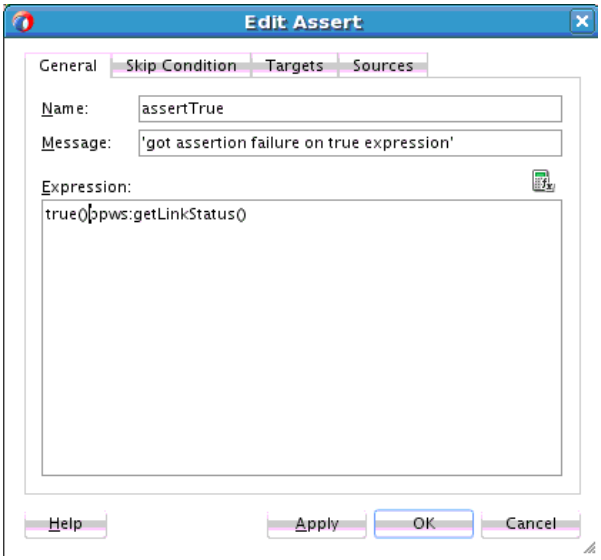


4. Expand the **Assert** activity.
5. To the right of the **Expression** field, click the **XPath Expression Builder** icon.
6. Create an expression.

7. When complete, click **OK**.

   The Assert dialog looks as shown in Figure 12-46.

   Figure 12-46 Assert Dialog



8. Click **Apply**, then **OK**.

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 the following example:

```
<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 *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

What Happens When You Create Assertion Conditions

The code segment in the `.bpel` file defines the specific operation after design completion.

For the following BPEL1.1 example, 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.

```
<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 the BPEL 1.1 example that follows, the `bpelx:assert` condition in the standalone assert activity, when evaluated to false, returns the following message:

```
got assertion failure on true expression
```

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.

```
<bpelx:assert expression="true()bpws:getLinkStatus()" message="'got assertion
failure on true expression'"
```

What You May Need to Know About Assertion Conditions

This section describes key assertion condition concepts.

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.

The following example shows multiple `bpelx:postAssert` extensions in a receive activity in BPEL 1.1:

```
      <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>
```

The following example shows multiple `bpelx:preAssert` extensions in an invoke activity in BPEL 1.1:

```
<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 How to Create Assertion Conditions.

Use of faultName and message Attributes

You can specify the `faultName` and `message` attributes of the `bpelx:postAssert` element, as shown in the schema definition in the following example for BPEL 1.1.

```
<invoke | receive | onMessage>
    standard-elements
    <bpelx:postAssert name="ncname"? expression="boolean-expr" faultName="QName"+
 message="generic-expr"+/> *
</invoke | receive | onMessage>
```

The following example shows the syntax for the `faultname` and `message` attributes.

```
<bpelx:postAssert name="Assert_2"
   message='multiple post assert Greater value fired'
   faultName="ns2:GreaterValue"
  expression="bpws:getVariableData('invar','payload','/ns1:process/ns1:input') <
 500"/>
```

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 is 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.

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. The following example provides details:

```
<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 the preceding example, the assertion with the expression that checks that the response credit rating is greater than zero is evaluated first. Table 12-11 describes the assertion behavior.

Table 12-11 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.

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. The following code provides several examples.

```
<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 management framework. For information, see Handling Faults with the Fault Management Framework.

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 runtime faults, see Introduction to the Business and Runtime Fault Categories of BPEL Faults.

For more information about fault policies, see Handling Faults with the Fault Management Framework.

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.

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. The following example provides details:

```
<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.

Assertion Conditions in a Standalone Assert Activity

You can also create assertion conditions in a standalone assert activity in a BPEL process service component. 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 about using the standalone assert activity, see How to Create Assertion Conditions.

What You May Need to Know About Postassertion and Preassertion Condition Schemas and Syntax

The assertion condition is specified as a nested extension element. The following example shows the postassertion condition schema definition in BPEL 2.0.

```
<invoke | receive | onMessage>
    standard-elements
    <bpelx:postAsserts>
       <bpelx:postAssert faultName="QName">
         <bpelx:expression expressionLanguage="anyURI"?>expression
         </bpelx:expression>
         <bpelx:message expressionLanguage="anyURI"?>expression</bpelx:message>
       </bpelx:postAssert>
   </bpelx:postAsserts>
</invoke | receive | onMessage>
```

The following example shows the postassertion condition syntax in BPEL 2.0.

```
<bpelx:postAsserts>
   <bpelx:postAssert faultName="ns2:InvalidInput">
     <bpelx:expression>number(concat($inputVariable.payload/client:input,'2')) <
      500</bpelx:expression>
      <bpelx:message>"AssertXpathPostInvoke_20 assert fired"</bpelx:message>
   </bpelx:postAssert>
</bpelx:postAsserts>
```

The following example shows the postassertion condition schema definition in BPEL 1.1. Note the differences between BPEL 1.1 and BPEL 2.0.

```
<invoke | receive | onMessage>
    standard-elements
    <bpelx:postAssert name="ncname" expression="boolean-expr" faultName="QName"+
 message="generic-expr"+/>
</invoke | receive | onMessage>
```

The following example shows the postassertion condition syntax in BPEL 1.1.

```
<bpelx:postAssert name="Assert_1"
   message='Post Invoke Multiple assert value fired'
   faultName="ns2:NegativeValue"
   expression="bpws:getVariableData('invar','payload','/ns1:process/ns1:input') >
0"/>
```

The following example shows the preassertion condition schema definition in BPEL 2.0.

```
<invoke | reply>
   standard-elements
   <bpelx:preAsserts>
     <bpelx:preAssert faultName="QName">
      <bpelx:expression expressionLanguage="anyURI"?>expression</bpelx:expression>
      <bpelx:message expressionLanguage="anyURI"?>expression</bpelx:message>
     </bpelx:preAssert>
   </bpelx:preAsserts>
</invoke | reply>
```

The following example shows the preassertion condition syntax in BPEL 2.0.

```
<bpelx:preAsserts>
    <bpelx:preAssert faultName="ns1:InvalidInput">
       <bpelx:expression>concat($inputVariable.payload/client:input,'2') >
       $inputVariable.payload/client:input</bpelx:expression>
       <bpelx:message>"AssertXpathPreInvoke_20 Assert test"</bpelx:message>
    </bpelx:preAssert>
</bpelx:preAsserts>
```

The following example shows the preassertion condition schema definition in BPEL 1.1. Note the differences between BPEL 1.1 and BPEL 2.0.

```
<invoke | reply>
    standard-elements
    <bpelx:preAssert name="NCName" expression="string" message="string"
    faultName="QName"/>
</invoke | reply>
```

The following example shows the preassertion condition syntax in BPEL 1.1.

```
<bpelx:preAssert name="Assert_1"
   expression="bpws:getVariableData('invar','payload','/ns1:process/ns1:input') >
   0"
   message='pre invoke assert NegativeInput fired'
   faultName="ns4:NegativeInput"/>
```

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 BPEL 1.1 invoke activity shown in the following example, if the XPath expression specified in the assertion condition returns false, the `NegativeCredit` fault is thrown.

```
<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.

Classifying SOAP Faults as Retriable

Starting with 12*c*, all web service SOAP faults are not automatically retried based on the fault code returned from the external service. SOAP faults are now retried only when the fault code is classified as server-related (also known as receiver-related). Fault codes classified as client-related do not result in retries. This differs from 11*g* Release 1 (11.1.1.*x*), in which Oracle SOA Suite retried all SOAP faults regardless of their fault code (all faults returned were converted to a `bpelx:remoteFault` in BPEL, which was retriable).

In 12*c* when a fault occurs in a reference binding component, the fault code is returned to a BPEL process. The fault is retried based on the setting in the fault code. This is

beneficial because you may want to retry the fault only under specific circumstances (such as a system downtime issue). For all other fault occurrences (such as incorrect input), you may not want a retry to occur. In fact, retries on all SOAP faults can delay the processing of legitimate messages.

As described in the *Simple Object Access Protocol (SOAP) 1.1* specification at `http://www.w3.org/TR/2000/NOTE-SOAP-20000508/#_Toc478383510`, a fault can have a code of server (also known as receiver) or client. The classification of faults determines whether the faults are retriable.

- Server

  Server errors indicate that the message cannot be processed for reasons not directly related to the message contents, but rather to the processing of the message. For example, processing can include communicating with a server that did not respond. The message may succeed at a later time. This is defined as a retriable fault.

- Client

  Client errors indicate that the message was incorrectly formed or did not contain the appropriate information to succeed. For example, the message may lack the proper authentication or payment information. This typically indicates that the message must first be changed before being resent. This is defined as a nonretriable fault.

This fault classification information is propagated into a `FabricInvocationException` error. For fault codes classified as client-related, the `retryType` flag within this exception is set to `NO_RETRY`.

If necessary, you can still invoke a retry on every fault. Set the `binding.ws` property `oracle.soa.always.retry.on.fault` to `true` in the `composite.xml` file. This enables Oracle SOA Suite to always retry on any SOAP faults regardless of the fault code.

```
<reference name="myreference"
. . .
<binding.ws port=". . ."
location=". . ."
<property name="oracle.soa.always.retry.on.fault">true</property>
</binding.ws>
```

13

Transaction and Fault Propagation Semantics in BPEL Processes

This chapter describes transaction and fault propagation semantics in Oracle BPEL Process Manager. It describes how to configure the transaction behavior for BPEL instances with initiating calls and the execution of one-way invocations. It also describes how to execute a business process without a transaction.

This chapter includes the following sections:

Introduction to Transaction Semantics

Transaction semantics in Release 12*c* 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.

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 Releases 11*g* and 12*c*, you only must set a new `transaction` property on the BPEL component being called (known as the callee process). You add `bpel.config.transaction` as follows:

- In the Create BPEL Process dialog for a new BPEL process.
- In the BPEL process service component section in the `composite.xml` file of an existing BPEL process (note the required prefix of `bpel.config.`).

This property configures the transaction behavior for BPEL instances with initiating calls. If you must change this setting later, you can use the Property Inspector.

The following example provides details:

```
<component name="InternalWareHouseService" version="2.0">
    <implementation.bpel src="BPEL/InternalWareHouseService.bpel"/>
    <property name="bpel.config.transaction" type="xs:string"
many="false">required | requiresNew | notSupported " </property>
  </component>
```

Table 13-1 describes the `required` (the default value) and `requiresNew` values and summarizes the behavior of the BPEL instance based on the settings.

Table 13-1 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 How to Add a BPEL Process Service Component and How to Define Deployment Descriptor Properties in the Property Inspector.

The following sections describe the transaction and fault behavior of setting `bpel.config.transaction` to either `required` or `requiresNew`.

BPELCaller Process Calls a BPELCallee Process That Has bpel.config.transaction Set to requiresNew

In Table 13-2, the BPELCaller process calls the BPELCallee process. The BPELCallee process has the property `bpel.config.transaction` set to `requiresNew`. Table 13-2 describes fault propagation and transaction behavior when `bpel.config.transaction` is set to this value.

Table 13-2 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.
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.

BPELCaller Process Calls a BPELCallee Process That Has bpel.config.transaction Set to required

In Table 13-3, the BPELCaller process calls the BPELCallee process. The BPELCallee process has the property `bpel.config.transaction` set to `required`. Table 13-3 describes fault propagation and transaction behavior when `bpel.config.transaction` is set to this value.

Table 13-3 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).

Introduction to Execution of One-way Invocations

A one-way invocation (with a possible callback) is typically exposed in a WSDL file as shown in the following example:

```
<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 execute 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` as follows:

- In the Create BPEL Process dialog for a new BPEL process.
- In the BPEL process service component section of the `composite.xml` file for an existing BPEL process.

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. With this setting, reliability is obtained with some performance impact on the database. In some cases, overall system performance can be impacted.
- `async.cache`: Incoming delivery messages are kept only in the in-memory cache. If performance is preferred over reliability, consider this setting. When set to `async.cache`, if the rate at which one-way messages arrive is much higher than the rate at which they are delivered, or if the server fails, messages can be lost. In addition, the system can become overloaded (messages become backlogged in the scheduled queue) and you can receive out-of-memory errors. Consult your own use case scenarios to determine if this setting is appropriate.

When you set `oneWayDeliveryPolicy` to `async.cache` in high availability environments, invoke and callback messages in the middle of execution at the time of a server crash may be lost or duplicated. Server failover is not supported for `async.cache`. For more information, see *High Availability Guide*.

- `sync`: Direct invocation occurs on the same thread. The scheduling of messages in the invoke queue is bypassed, and the BPEL instance is invoked synchronously. In some cases this setting can improve database performance.

For more information about setting the `bpel.config.oneWayDeliveryPolicy` property, see How to Add a BPEL Process Service Component and How to Define Deployment Descriptor Properties in the Property Inspector.

Table 13-4 describes the behavior when the main process calls the subprocess asynchronously. Table 13-4 is based on the use cases described in BPELCaller Process Calls a BPELCallee Process That Has bpel.config.transaction Set to requiresNew and BPELCaller Process Calls a BPELCallee Process That Has bpel.config.transaction Set to required.

Table 13-4 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.
`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.
`onewayDeliveryPolicy=async.cache` and `transaction=requiresNew` or `transaction=required`	The BPELCaller does not get a response because the caller thread returns before the request is handled. The BPELCallee transaction is rolled back if the fault is not handled. The message is lost because it is not saved in the database.	The BPELCaller does not get a response because the caller thread returns before the request is handled. The BPELCallee transaction is rolled back if the fault is not handled. The message is lost because it is not saved in the database.

Executing a Business Process Without a Transaction

You can execute a business process without the need for a transaction. A transaction is only used at the following points in the process execution:

- At the dehydration point when the internal processing state must be stored in the back end data store.
- When storing the audit trail or instance tracking-related data during process execution.

When Should I Use a BPEL Process Without a Transaction?

Executing a business process without a transaction is beneficial in scenarios similar to the following:

- Assume you have a BPEL process in which a flowN activity spawns 2000 branches. Each branch invokes a remote synchronous web service that takes 500 ms to respond. Because the BPEL process service engine executes flowN branches individually in a single thread, processing all 2000 branches with each one invoking a synchronous web service takes close to 1000 seconds and the instance does not have access to the dehydration point during this processing. The transaction can extend for 1000 seconds and can time out (the default transaction timeout setting is 300 seconds). Everything can be performed directly in memory without the need for a transaction.
- The duration of a transaction gets tied up with the life cycle of business process execution. For example, assume an asynchronous BPEL process includes a receive activity followed by an assign activity in which a complex XSL transformation performed on a large document takes 30 seconds. This is followed by a callback to the client. If executed in a transaction, the BPEL process service engine starts the transaction at the receive activity and holds a lock inside the database on the instance while the instance is executing.

  As an alternative, all activities can be performed in memory and discarded if an error occurs. A transaction is not required because a database update does not occur during instance execution. A transaction is only required once instance execution reaches the dehydration point, where the BPEL process service engine updates the instance state, and so on.
- Assume a BPEL process invokes another service or partner link that is synchronous and participates in a BPEL process service engine's JTA transaction (for example, if a BPEL process invokes the `TaskServiceBean`, which has `TransactionAttribute=REQUIRED`, and there is a `TaskServiceBean` time out and the transaction is roll backed). Even the BPEL process service engine's JTA transaction gets rolled back and the BPEL process is unable to handle the error from the `TaskServiceBean`.
- If a business process invokes a synchronous service and that service is performing complex work that takes a considerable amount of time, the BPEL process service engine transaction can time out. Even though the synchronous service is performing correctly, the BPEL process service engine rolls back once the business process gets a response from a remote service.

Guidelines for Executing Without a Transaction

To execute a business process without a transaction, select **notSupported** from the **Transaction** list when creating a BPEL process in the Create BPEL Process dialog.

When set, the following behavior occurs:

- All XA distributed transaction benefits are disabled.

  When a business process is configured to run in non-transactional mode, the instance execution is not wrapped in an XA transaction, resulting in potential duplicate instances, but no loss of message(s). As there is no overhead of a transaction, the non-transactional mode provides better performance. You can use the non-transactional option where duplicate instances are acceptable.

- The business process cannot invoke any partner that expects to participate in a transaction (that is, the partner has the `TransactionAttribute` set to `MANDATORY`).

- The invoke from the business process is fire and forget (that is, once the invoke is finished, it is delivered to the partner. Even if the invoker's transaction rolls back afterwards, the invoke message is not rolled back).

Even with `bpel.config.transaction` set to `notSupported`, the dehydration point starts a transaction to save the internal BPEL process engine state into the back end. This means the dehydration concept still applies for the business process. This feature only guarantees that business process activities such as an assign, an invoke, and others are executed without a transaction.

This property configures the transaction behavior of a BPEL instance in the case of initiating calls. Table 13-5 describes the behavior of the BPEL instance based on the `bpel.config.transaction` property setting.

Table 13-5 BPEL Process Instance Behavior Based on transaction Property Settings

Transaction Type	transaction = requiresNew	transaction = required	transaction = notSupported
Request/response (initiating)	A new transaction is created for the execution. The existing transaction (if there is one) is suspended.	The process joins a caller's transaction (if there is one) or creates a new transaction (if there is not a transaction).	Business process activities are executed without a transaction. The transaction is only used to save internal service engine/ instance state and audit details. Any `bpelx:rollback` fault is *not* propagated back to the client because the current instance does not participate in the client's transaction.
One-way (initiating, `bpel.config.oneWayDeliveryPolicy=sync`)	A new transaction is created for the execution and the existing transaction (if there is one) is suspended.	The invoke message is processed using the same thread in the same transaction.	Business process activities are executed without a transaction. A transaction is only used to save internal service engine/ instance state and audit details. Any `bpelx:rollback` fault is not propagated back to the client.

Table 13-5 (Cont.) BPEL Process Instance Behavior Based on transaction Property Settings

Transaction Type	transaction = requiresNew	transaction = required	transaction = notSupported
One-way asynchronous	Not applicable.	Not applicable.	Business process activities are executed without a transaction. The transaction is only used to save internal service engine/ instance state and audit details.

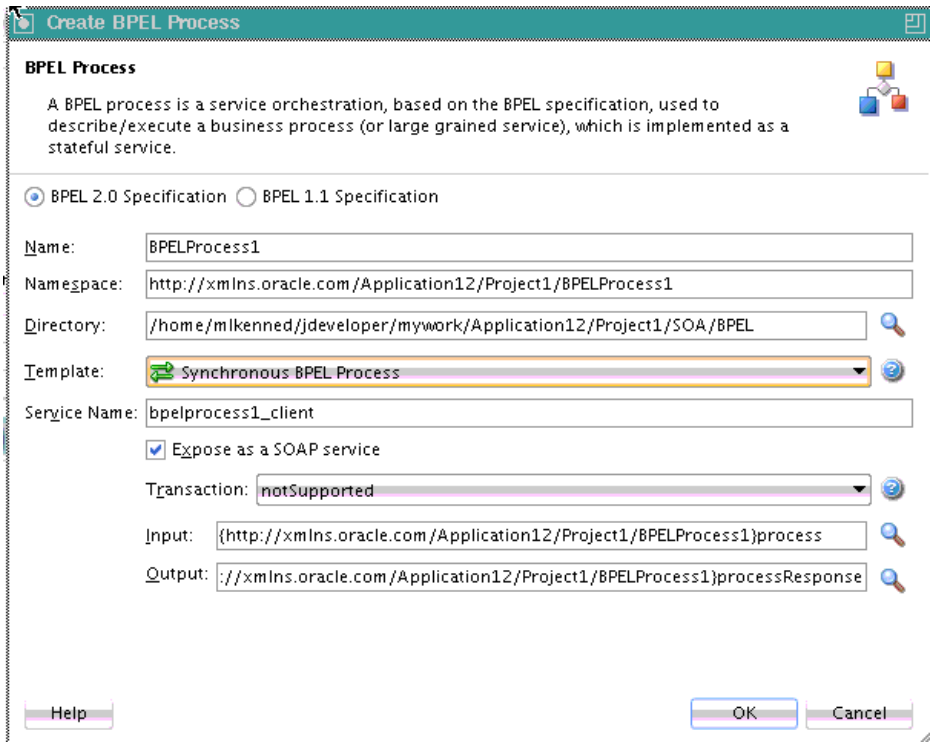
How to Create a Synchronous BPEL Process Without a Transaction

You can create a synchronous BPEL process without a transaction in the Create BPEL Process dialog.

To create a synchronous BPEL process without a transaction:

1. Create a BPEL process service component in the SOA composite application, as described in How to Add a BPEL Process Service Component.
2. From the **Template** list, select **Synchronous BPEL Process**.
3. From the **Transaction** list, select **notSupported**. Figure 13-1 provides details.

Figure 13-1 Create BPEL Process Dialog



4. Click **OK**.

How to Create an Asynchronous BPEL Process Without a Transaction

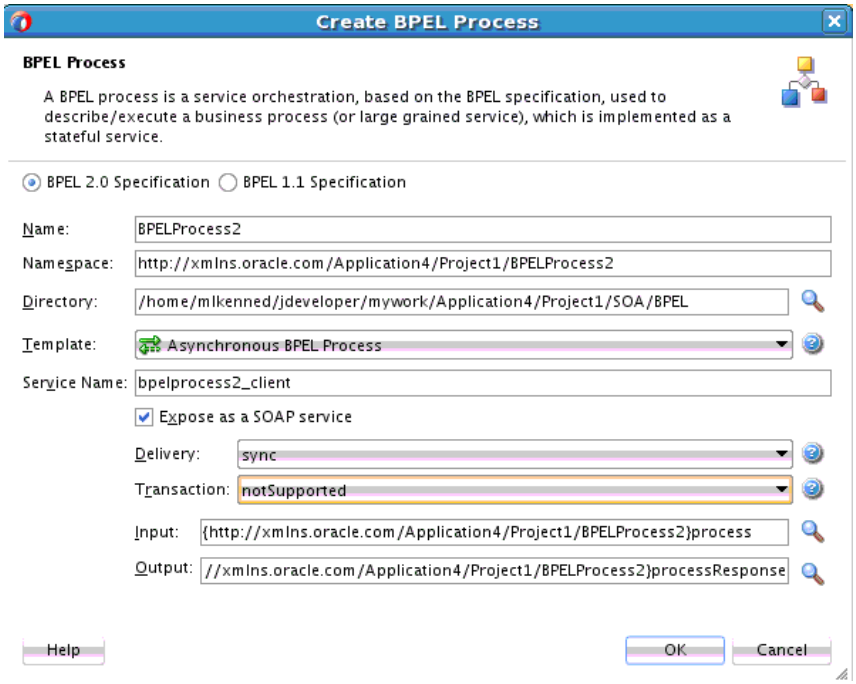
You can create an asynchronous BPEL process without a transaction in the Create BPEL Process dialog.

To create an asynchronous BPEL process without a transaction:

1. Create a BPEL process service component in the SOA composite application, as described in How to Add a BPEL Process Service Component.
2. From the **Template** list, select **Asynchronous BPEL Process**.
3. From the **Delivery** list, select **sync**.

   The dialog is refreshed to display the **Transaction** list.
4. From the **Transaction** list, select **notSupported**. Figure 13-2 provides details.

Figure 13-2 Create BPEL Process Dialog



5. Click **OK**.

14

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 of SOA composite applications. It describes how to add custom classes and JAR files, use the Java embedding activity, embed service data objects (SDOs) with `bpelx:exec`, and implement a custom Connection Manager class with a BPEL process.

This chapter includes the following sections:

You can also invoke a spring component. For more information, see Integrating the Spring Framework in SOA Composite Applications.

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 code that can perform the necessary function, and you want to use the existing code rather than start over with BPEL.

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

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.

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 are completed).

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 the following code to embed the `invokeSessionBean` Java bean:

```
<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>
```

How to Embed Java Code Snippets in a BPEL 2.0 Process

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. The following example provides details.

```
<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.

The following example shows the import syntax for BPEL 2.0:

```
<import location="class/package name"
 importType="http://schemas.oracle.com/bpel/extension/java"/>
```

Note:

The BPEL 2.0 import syntax differs from BPEL 1.1, which uses the following syntax:

```
<bpelx:exec import="class/package name"/>
```

The following example shows a BPEL file with two Java embedding activities for a project that supports BPEL 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 about using this activity, see Using Java Embedding in a BPEL Process in .

How to Use an XML Facade to Simplify DOM Manipulation

You can use an XML facade to simplify DOM manipulation. 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. The following provides an example:

```
<bpelx:exec name= ...
   <![CDATA
    ...
   Element element = ...
        (Element)getVariableData("input","payload","/loanApplication/"):
   //Create an XMLFacade for the Loan Application Document
   LoanApplication xmlLoanApp=
        LoanApplicationFactory.createFacade(element);
...
```

How to Use bpelx:exec Built-in Methods

Table 14-1 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 14-1 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

Table 14-1 (Cont.) Built in Methods for bpelx:exec

Method Name	Description
`String setStatus( String status ) / String getStatus()`	Status of this instance
`void setCompositeInstanceTitle(String title)`	Sets the composite instance title
`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 a file stored in the archive
`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

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 composite 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 55.a.

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 composite 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 *Developing Fusion Web Applications with Oracle Application Development Framework*.

For more information on invoking a SOA composite application, see *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

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.

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.

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 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 or a semicolon (`;`) on Windows.

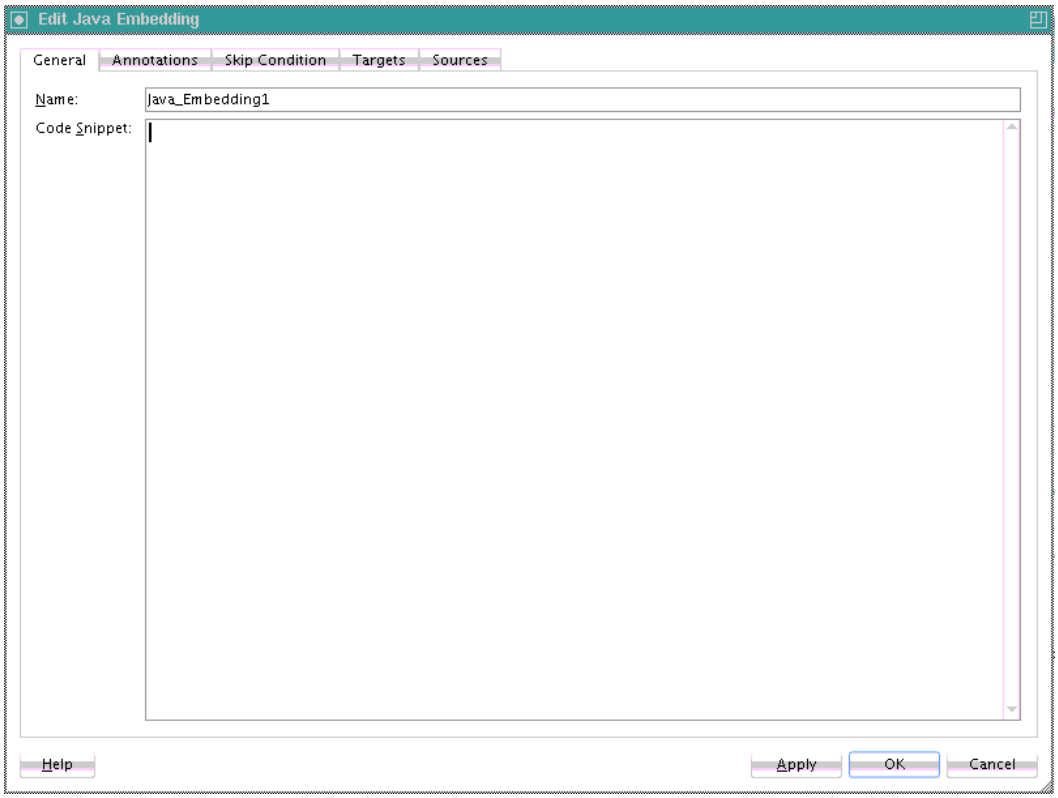
How To Use Java Embedding in a BPEL Process in Oracle JDeveloper

To use Java embedding in a BPEL process in Oracle JDeveloper:

1. From the Components window, expand **Oracle Extensions**.
2. Drag the **Java Embedding** activity into the designer.
3. Click the **Java Embedding** activity to display its property fields in the Property Inspector or double-click the **Java Embedding** activity to display the Java Embedding dialog.

   For information about editing activities in the Property Inspector, see How to Edit BPEL Activities in the Property Inspector.
4. In the **Name** field, enter a name.
5. In the **Code Snippet** field, enter (or cut and paste) the Java code. Figure 14-1 provides details.

Figure 14-1 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.

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.

Embedding Service Data Objects with bpelx:exec

You can embed SDO code in the `.bpel` file with the `bpelx:exec` tag. In the syntax provided in the following example, `mytest.apps.SDOHelper` is a Java class that modifies SDOs.

```
</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>
```

The following provides an example of the Java classes `modifySDO(o)` and `print(o)` that are embedded in the BPEL file:

```
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();
}          }
```

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.

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.

15

Using Events and Timeouts in BPEL Processes

This chapter describes how to use events and timeouts. It describes how to create a pick activity to select to continue a process or wait, set timeouts for request-response operations on receive activities, create wait activities to set an expiration time, create OnEvent branches in BPEL 2.0 to wait for message arrival, set timeouts on synchronous processes, and invoke an Oracle Enterprise Scheduler job in a BPEL process.

This chapter includes the following sections:

Introduction to Event and Timeout Concepts

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.

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.

Selecting Between Continuing or Waiting on a Process with a Pick Activity

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 15-1 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` is for seconds

     – `M` for one minute

     – `H` is for hour

     – `D` is for day

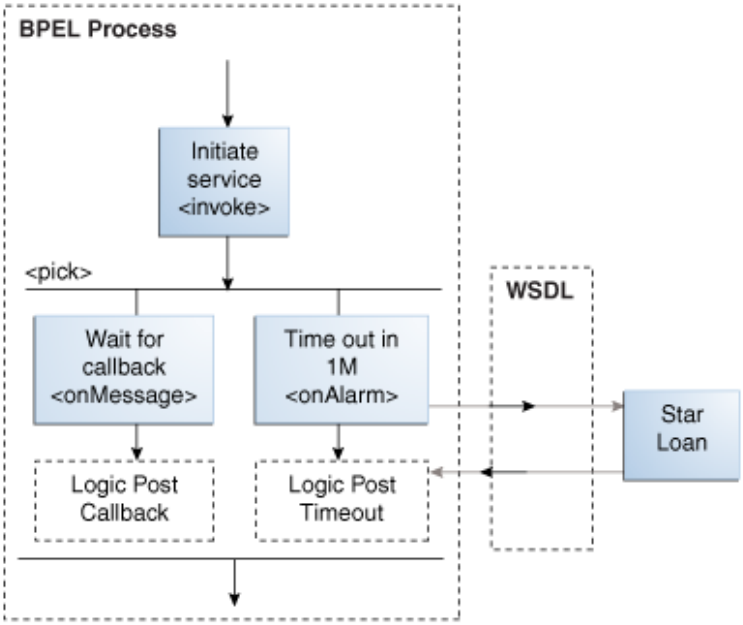
     – `Y` is for year

     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 is not executed.

Figure 15-1 Overview of the Pick Activity



An onMessage branch is similar to a receive activity in that it receives messages. 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** check box), 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.

How To Create a Pick Activity

To create a pick activity:

1. In the , double-click the BPEL process service component.
2. In the Components window, expand **BPEL Constructs** > **Structured Activities**.
3. Drag a **Pick** activity into the designer.

   The **Pick** activity includes an **OnMessage** branch. Figure 15-2 provides an example.

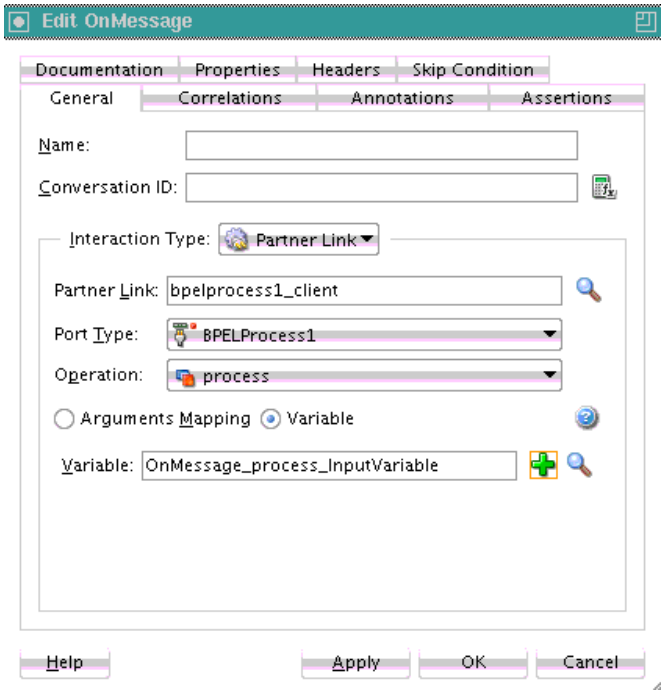
Figure 15-2 Pick Activity

4. Click the **OnMessage** branch to display its property fields in the Property Inspector or double-click the **OnMessage** branch.

   For information about editing activities in the Property Inspector, see How to Edit BPEL Activities in the Property Inspector.

5. Edit its attributes to receive the response from the loan service. Figure 15-3 provides an example.

Figure 15-3 OnMessage Branch

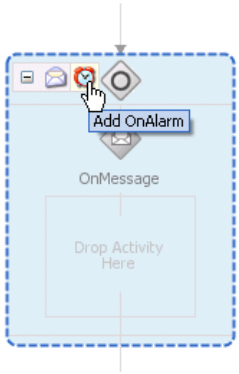


6. Select the **Pick** activity.

   Icons for adding additional **OnMessage** branches and an **OnAlarm** branch are displayed.

7. Click **Add OnAlarm**, as shown in Figure 15-4.

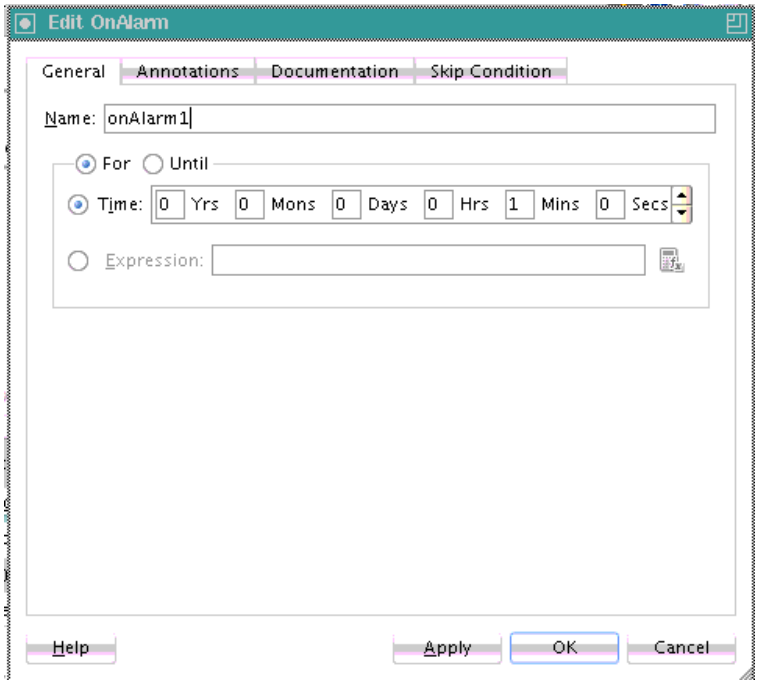
Figure 15-4 onAlarm Branch Creation



An **OnAlarm** branch is displayed.

8. Double-click the **OnAlarm** branch of the **pick** activity and set its time limit to `1` minute. Figure 15-5 provides an example.

Figure 15-5 OnAlarm Branch



9. Click **OK**.

What Happens When You Create a Pick Activity

The code segment in the following example defines the pick activity for this operation after design completion:

```
<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>
```

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.

Setting Timeouts for Request-Reply and In-Only Operations in Receive Activities

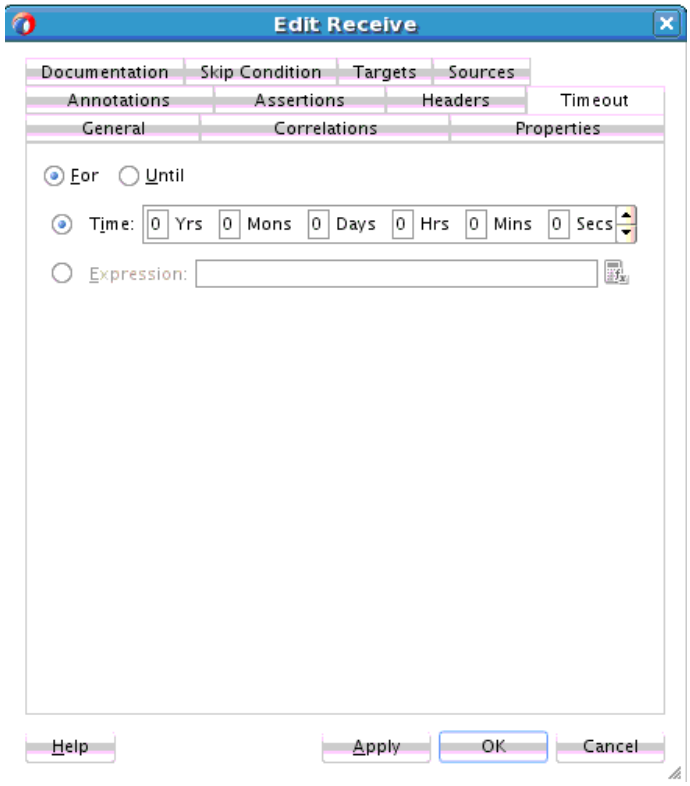
You can provide a timeout setting for the following types of operations in BPEL versions 1.1 and 2.0:

- Request-reply (synchronous) operations.
- In-only receive (asynchronous) operations. In this scenario, the receive activity must be a midprocess activity and not the activity that creates a new instance (that is, the **Create Instance** check box in the Receive dialog is selected).

This provides an alternative to using the onMessage and onAlarm branches of a pick activity to specify a timeout duration for partner callbacks.

Figure 15-6 shows the **Timeout** tab of a midprocess receive activity in which you set a timeout.

Figure 15-6 Timeout Tab of a Receive Activity



For information about key concepts to understand before setting timeouts for request-reply and in-only operations in receive activities, see What You May Need to Know About Setting Timeouts for Request-Reply and In-Only Operations.

For information about how to set a timeout in a receive activity in Oracle JDeveloper, see How to Set Timeouts in Receive Activities.

How to Set Timeouts in Receive Activities

Set timeouts in the following scenarios:

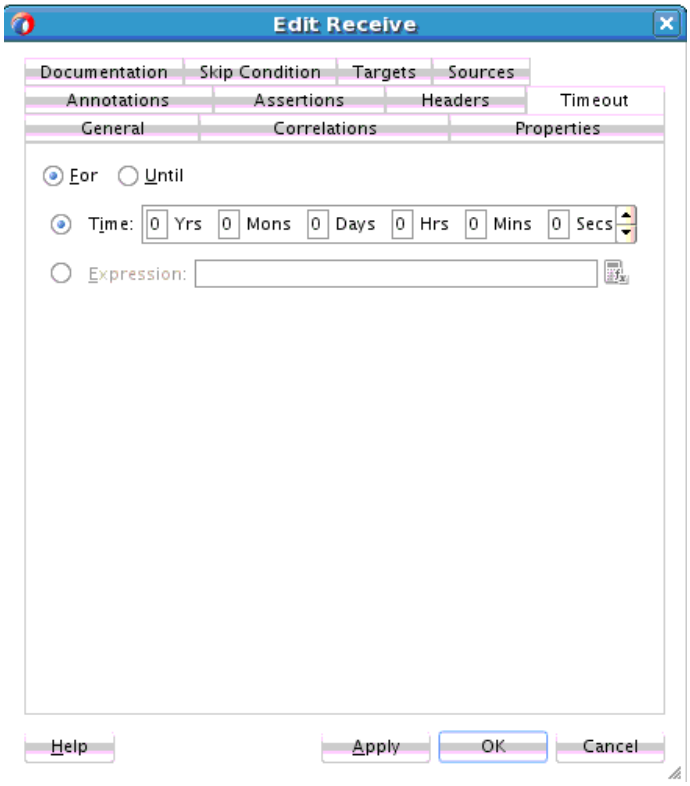
- The **Create Instance** check box is deselected.
- The receive activity is in the middle of the BPEL process (in most cases)

To set timeouts in receive activities:

1. In the , double-click the BPEL process service component.
2. In the Components window, 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 15-7.

Figure 15-7 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**.

What Happens When You Set Timeouts 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 the following example:

```
<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 the following code:

```
<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 the following code can display:

```
<bpelx:for="bpws:getVariableData('inputVariable','payload','/tns:waitValue/tns:for
')"/>
```

What You May Need to Know About Setting Timeouts for Request-Reply and In-Only Operations

The following sections describe request-reply and in-only timeout operations 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

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 the following example for BPEL 1.1.

```
<receive | bpelx:for="duration-expr">
    standard-elements
</receive>
```

For BPEL 2.0, the syntax is as shown in the following example:

```
<receive | <bpelx:for>'duration-expr'</bpelx:for>
    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.

Timeout Settings as an Absolute Date Time

You can specify a timeout setting as an absolute deadline for request-response receive activities. For BPEL 2.0, the syntax is as shown in the following example:

```
<receive <bpelx:until>"deadline-expr"</bpelx:until>
</receive>
```

For BPEL 1.1, the syntax is as shown in the following example:

```
<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.

Timeout Settings Computed Dynamically with an XPath Expression

The timeout setting for request-response receives, in-only receives (callback), 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

The following example shows the syntax for using XPath expressions in BPEL 1.1.

```
<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.

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 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.

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` 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.)

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.

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.

Setting an Expiration Time with a Wait Activity

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.

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.

To specify the minimum wait time:

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**.

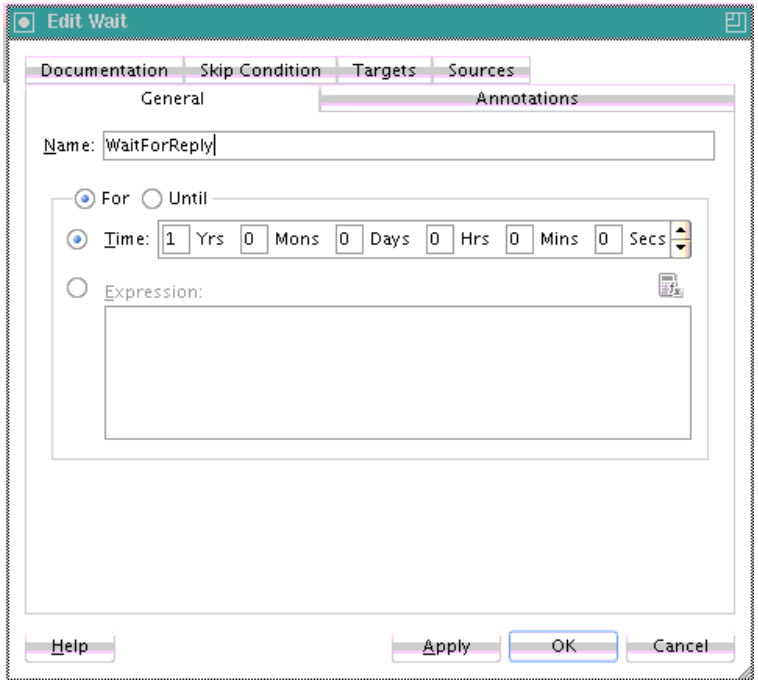
4. In the **Value** field, specify a value in seconds.
5. Click **Apply**.
6. Click **Return**.

How to Create a Wait Activity

To create a wait activity:

1. In the Components window, 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 15-8.

Figure 15-8 Wait Dialog



What Happens When You Create a Wait Activity

Exactly one of the expiration criteria must be specified, as shown in the following example for BPEL 2.0.

```
<wait <for>'duration-expr'</for> | <until>'duration-expr'</until>
    standard-elements
  </wait>
```

The following example shows the BPEL 1.1 syntax.

```
<wait (for="duration-expr" | until="deadline-expr") standard-attributes>
    standard-elements
  </wait>
```

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 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.

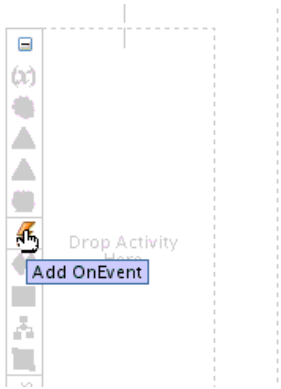
The onEvent event handlers are associated with an enclosed scope. The onEvent event handlers are enabled when their scope is initialized and disabled when their scope ends. When enabled, any number of events can occur. They are processed in parallel to the scope's primary activity and in parallel to each other. Message events also represent service 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.

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 15-9.

   Figure 15-9 Add OnEvent Icon

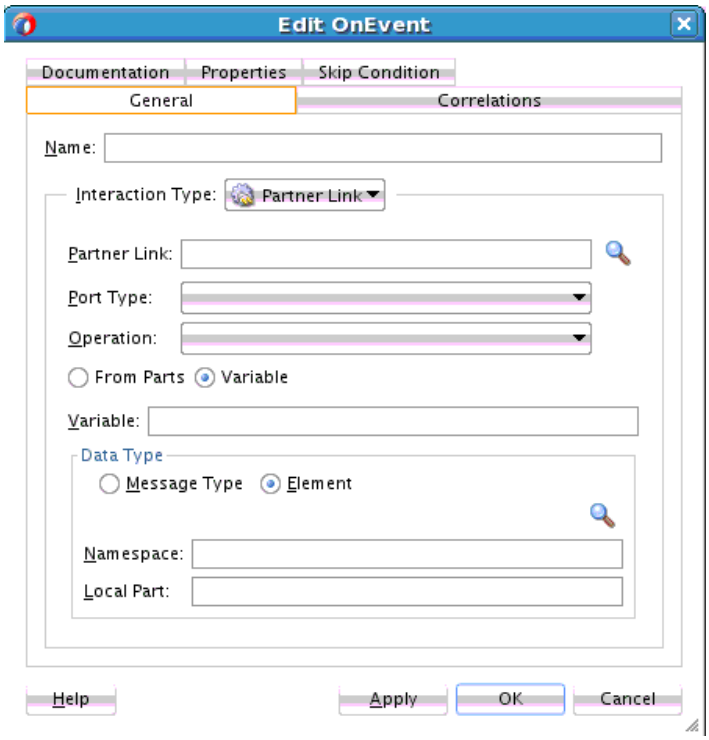
   

   This creates an **OnEvent** branch and an enclosed scope activity.

2. Double-click the **OnEvent** branch.

   The OnEvent dialog is displayed, as shown in Figure 15-10.

Figure 15-10 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** element.

5. Click **Apply**, then click **OK**.

6. Continue the design of your BPEL process.

What Happens When You Create an OnEvent Branch

The following example 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.

```
<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>
```

Setting Timeouts for Durable Synchronous Processes

For durable 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 Specifying Transaction Timeout Values in Durable Synchronous Processes.

Invoking an Oracle Enterprise Scheduler Job in a BPEL Process

You can invoke an Oracle Enterprise Scheduler job in a BPEL process. An Oracle Enterprise Scheduler job is a unit of work in the form of either Java, a database stored procedure, or any executable. A job definition is associated with Oracle Enterprise Scheduler, which describes how to execute the job. An Oracle Enterprise Scheduler web service submits the job from within a BPEL process and associates a schedule with that job request.

The scheduled Oracle Enterprise Scheduler job resides in a runtime environment and is accessible with an Oracle Metadata Services Repository (MDS Repository) connection, using database-based access.

Note:

This section describes how to submit a job from a BPEL process, and not how to wait for the job to complete. If you want the BPEL process to wait for the job to complete, you must invoke the web service to request a callback when the job completes and then perform a receive to get the callback. For more information, see Chapter "Using the Oracle Enterprise Scheduler Web Service" of *Developing Applications for Oracle Enterprise Scheduler*.

How to Create Oracle Database and SOA-MDS Connections

To create Oracle database and SOA-MDS connections:

1. Create a SOA composite application. For information, see Creating a SOA Application.
2. Create a BPEL process in the SOA Composite Editor (for this example, a synchronous BPEL process is created). For information, see How to Add a BPEL Process Service Component.
3. Double-click the BPEL process in the SOA Composite Editor.

   Oracle BPEL Designer is displayed.
4. Create an Oracle database connection. This is required for querying Oracle Enterprise Scheduler jobs.

   a. From the **File** main menu, select **New** > **Application**.

b. From the **Categories** list, select **Connection**.

c. Select **Database Connection**.

The Create Database Connection wizard is displayed.

d. Complete the dialogs of the Create Database Connection wizard to create the connection to the Scheduler Oracle Metadata Services Repository database for the runtime server where Oracle Enterprise Scheduler is deployed, and click **Finish**.

5. Create a SOA-MDS connection. A database-based MDS Repository is used for retrieving the jobs to select.

a. From the **File** main menu, select **New** > **Application**.

b. From the **Categories** list, select **Connection**.

c. Select **SOA-MDS Connection**.

The Create SOA-MDS Connection dialog is displayed.

d. From the **Connection Type** list, select **DB Based MDS**.

e. From the **Connection** list, ensure that the database connection created in Step 4 is displayed.

f. From the **Select MDS partition** list, select the partition that includes Oracle Enterprise Scheduler jobs. For jobs defined in the Oracle Enterprise Scheduler predeployed native hosting application, the MDS partition name is **essUserMetadata**.

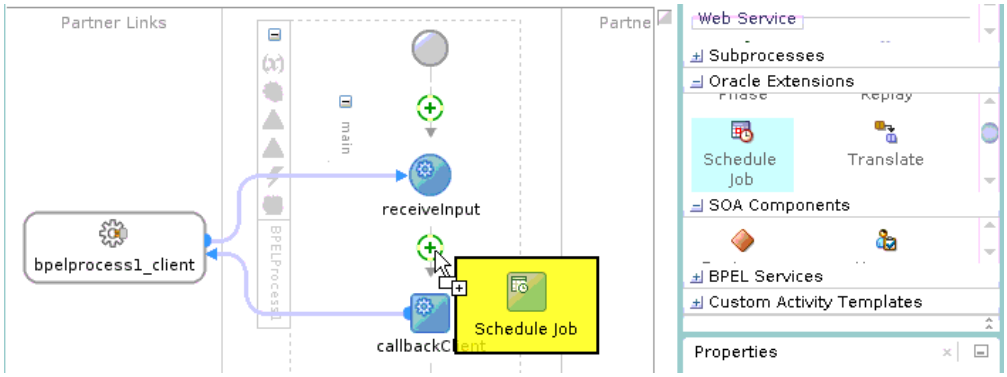
g. Complete the remaining fields of the dialog to create the SOA-MDS connection, and click **OK**.

How to Create a Schedule Job Activity

To create a schedule job activity:

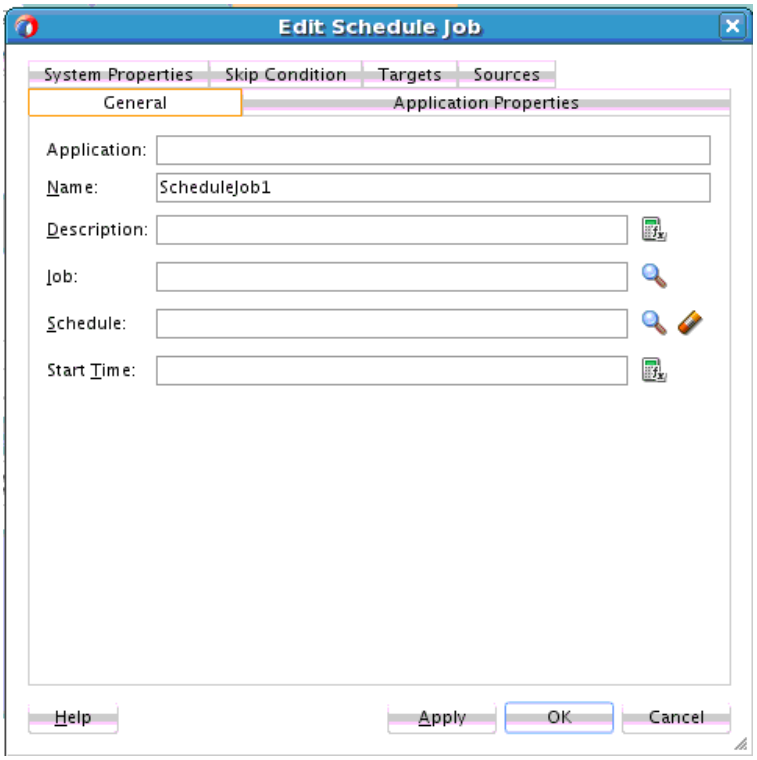
1. From the Components window, expand **Oracle Extensions**.
2. Drag a **Schedule Job** activity into the BPEL process, as shown in Figure 15-11.

Figure 15-11 Schedule Job Icon



3. Double-click the activity to invoke the Edit Schedule Job dialog. Figure 15-12 provides details. This dialog enables you to specify the application, the description, the Oracle Enterprise Scheduler job, the job schedule, and the job start time.

Figure 15-12 Edit Schedule Job Dialog - General Tab



4. Provide values appropriate to your environment, as described in Table 15-1, and click **OK**,

Table 15-1 Edit Schedule Job Dialog - General Tab

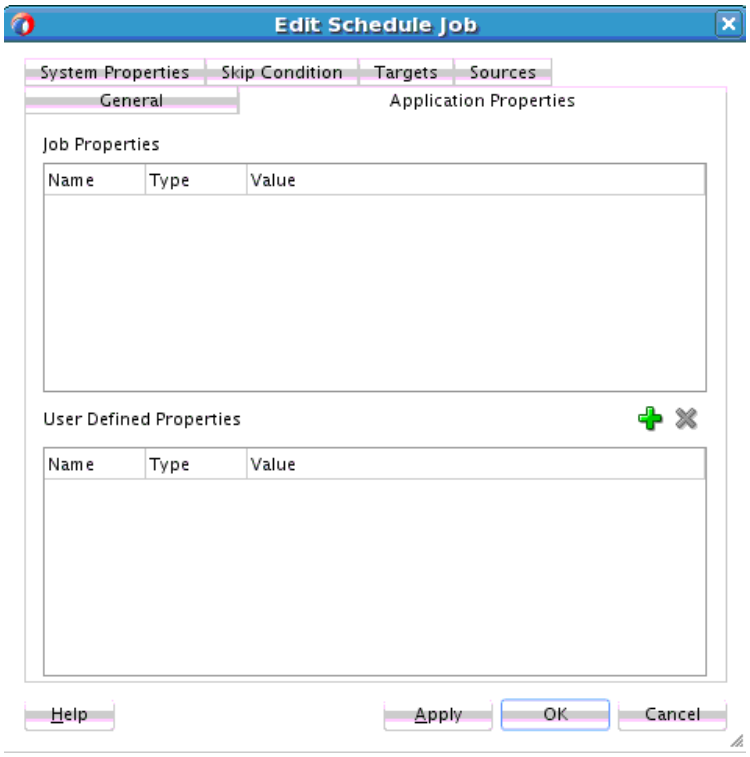
Field	Description
Application	Displays the value of the selected job's **SYS_effectiveApplication** property. This property *must* be set, or an error message is displayed and you cannot proceed. The editable state of this field depends on the selected job definition: • If the selected job definition provides **SYS_effectiveApplication**, then the value for this property is displayed and this field is not editable. • If the job definition does not provide **SYS_effectiveApplication**, then this field is editable and you must specify the application name in the **User Defined Properties** section of the **System Properties** tab.
Name	Specify the name of the job.
Description	Specify a description for the request.

Table 15-1 (Cont.) Edit Schedule Job Dialog - General Tab

Field	Description
Job	Click the **Search** icon to invoke the Enterprise Scheduler Browser dialog to select the job from the SOA-MDS connection. When you select a job, any system or application properties defined for that job are displayed in the **Application Properties** and **System Properties** tabs.
Schedule	Click the **Search** icon to invoke the Enterprise Scheduler Browser dialog to select the job schedule. If not specified, the job is executed immediately. You define schedules in Oracle Enterprise Manager Fusion Middleware Control. Those schedules are then displayed for selection in the Enterprise Scheduler Browser dialog. For more information, see "Creating or Editing Predefined Job Schedules" of *Administering Oracle Enterprise Scheduler*.
Start Time	Click the XPath Expression Builder icon to specify the start time as an XPath expression. The start is separate from the schedule, and indicates when the job takes effect. If a start time is not specified, the start time is immediate.
End Time	Click the XPath Expression Builder icon to specify the end time as an XPath expression. The end is separate from the schedule, and indicates when the job ends. If a schedule is not specified, this field is not displayed.

5. Click the **Application Properties** tab. Application properties are unique to a specific job. When you select an Oracle Enterprise Scheduler job in the Edit Schedule Job dialog - **General** tab, the application properties defined in the job are displayed in this dialog. You can also specify your own application properties in the **User Defined Properties** section. Figure 15-13 provides details.

Figure 15-13 Edit Schedule Job Dialog - Application Properties Tab



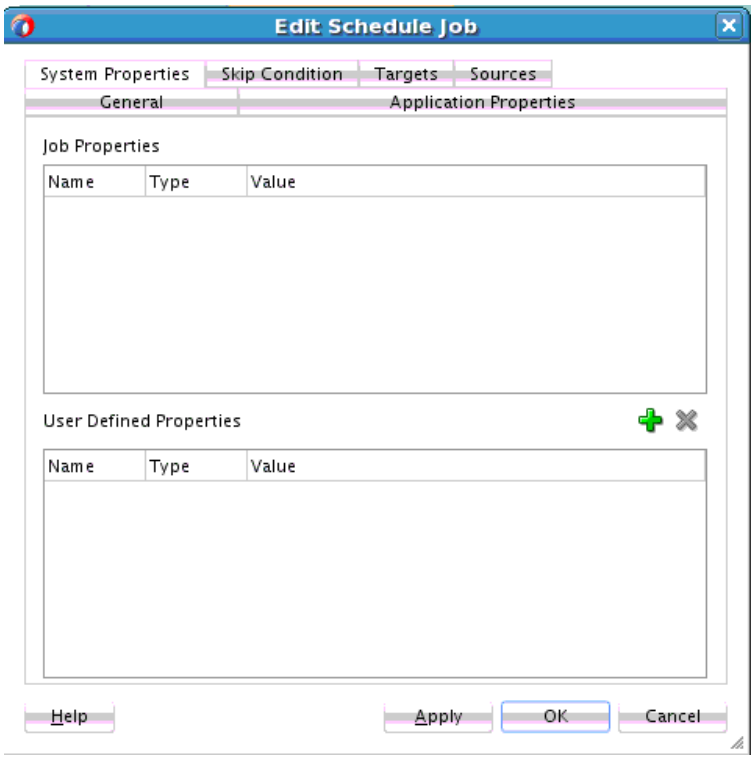
6. Provide values appropriate to your environment, as described in Table 15-2, and click **OK**.

Table 15-2 Edit Schedule Job Dialog - Application Properties Tab

Field	Description
Job Properties	Displays the application properties defined by the job. Only the values can be modified. The properties in this table cannot be removed. Double-click a property to edit its value or click the **Browse** icon to the right of the **Value** field to specify an XPath expression.
User-Defined Properties	Displays the application properties that you have added for this request. You can add, modify, and remove properties in this table.

7. Click the **System Properties** tab. System properties are parameters with names reserved by Oracle Enterprise Scheduler. Oracle Enterprise Scheduler represents parameter names that are known and used by the system in the `SystemProperty` class. When you select an Oracle Enterprise Scheduler job in the Edit Schedule Job dialog - **General** tab, the system properties defined in the job are displayed in this dialog. You can also specify your own system properties in the **User Defined Properties** section. Figure 15-14 provides details.

Figure 15-14 Edit Schedule Job Dialog - System Properties Tab



For more information about system properties, see Chapter "Using Parameters and System Properties" of *Developing Applications for Oracle Enterprise Scheduler*.

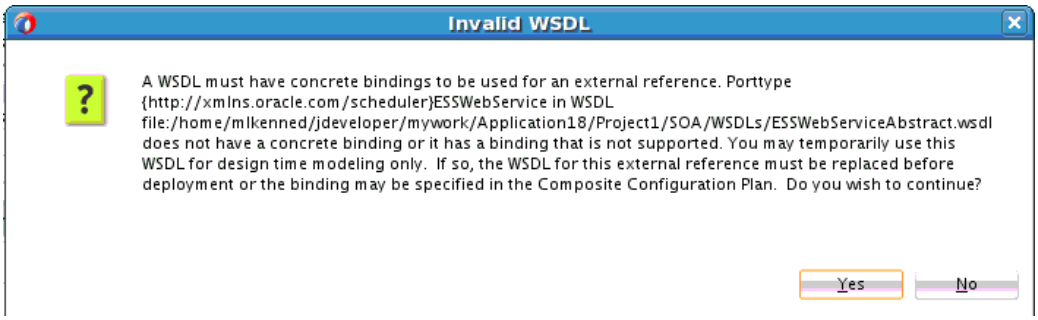
8. Provide values appropriate to your environment, as described in Table 15-3, and click **OK** to complete configuration.

Table 15-3 Edit Schedule Job Dialog - System Properties Tab

Field	Description
Job Properties	Displays the system properties defined by the job. Only the values can be modified. Double-click a property to edit its value or click the **Browse** icon to specify an XPath expression in the Expression Builder dialog.
User-Defined Properties	Displays the system properties that you have added for this request. You can add, modify, and remove properties in this table. Select from a fixed list of system property names in this table.

The message shown in Figure 15-15 is displayed because the Oracle Enterprise Scheduler web service includes an abstract WSDL.

Figure 15-15 WSDL Message



A BPEL process requires the following:

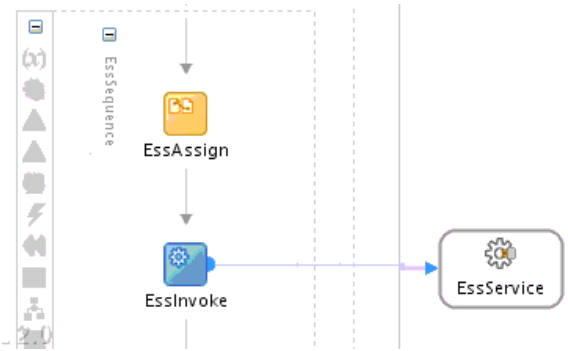
- A concrete WSDL
- A WSDL with partner links

9. Click **Yes**.

   A concrete wrapper WSDL is created for the abstract WSDL. The wrapper WSDL includes an Oracle Enterprise Scheduler partner link that is added to the BPEL process.

10. Expand the schedule job activity in the BPEL process to display its contents. Figure 15-16 provides details.

Figure 15-16 Expanded Job Schedule Activity in a BPEL Process.



The expanded schedule job activity consists of the following automatically configured activities:

- **EssAssign** activity: Contains copy rules operations for the system and application properties and other job information.
- **EssInvoke** activity: Invokes the Oracle Enterprise Scheduler partner link.
- **EssService** activity: Contains the Oracle Enterprise Scheduler web service partner link.

11. Go to the SOA composite application in the SOA Composite Editor.
12. In the **External References** swim lane, double-click the **EssService** partner link.

    The Update Reference dialog is displayed.

13. In the **WSDL URL** field, specify a concrete WSDL for the reference binding component, and click **OK**.

How to Attach Security Policies to the Service and Reference Binding Components

To attach security policies to the service and reference binding components

1. Right-click the **EssService** reference binding component, and select **Configure SOA WS Policies** > **For Request**.

   The Configure SOA WS Policies dialog is displayed.

2. In the **Security** section, click the **Add** icon.
3. Select **oracle/wss_username_token_client_policy**, and click **OK**.
4. In the Configure SOA WS Policies dialog, click **OK**.
5. Right-click the service binding component, and select **Configure SOA WS Policies**.

   The Configure SOA WS Policies dialog is displayed.

6. In the **Security** section, click the **Add** icon.
7. Select **oracle/wss_username_token_service_policy**, and click **OK**.

   Design is now complete.

Note:

The Oracle Enterprise Scheduler web service is by default not secure. You must first secure it with an Oracle Web Services Manager policy using a WLST command or Oracle Enterprise Manager Fusion Middleware Control before using that web service to submit a job from a BPEL process.

16

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:

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.

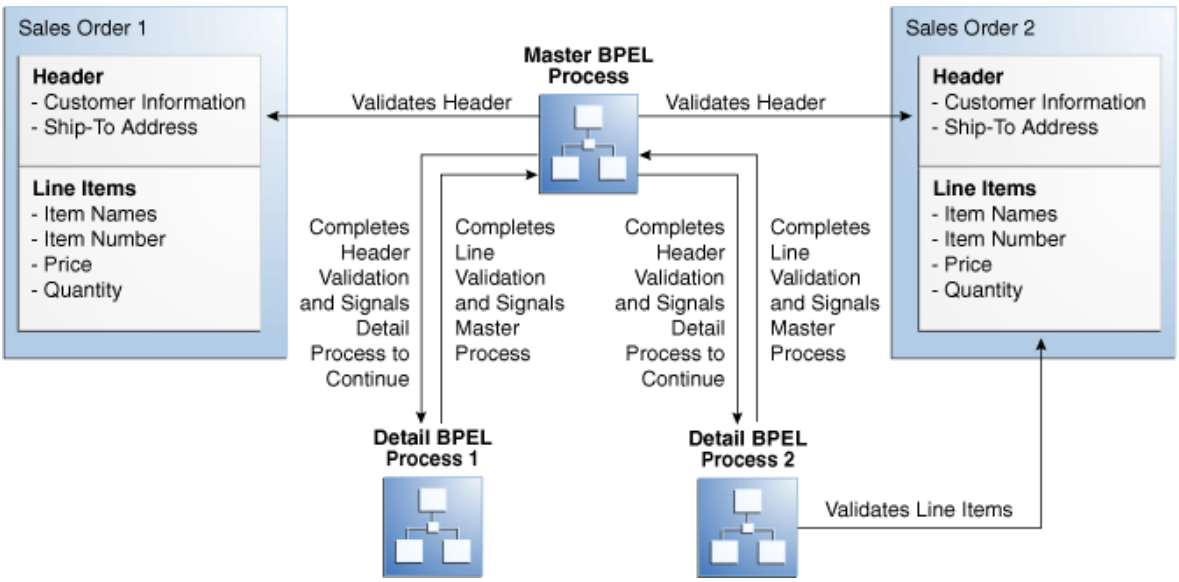
Potential coordination points are as follows:

- 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 16-1 provides an overview of the header and line item validation coordination points between one master process and two detail processes.

Figure 16-1 Master and Detail Coordination Overview (One BPEL Process to Two Detail Processes)



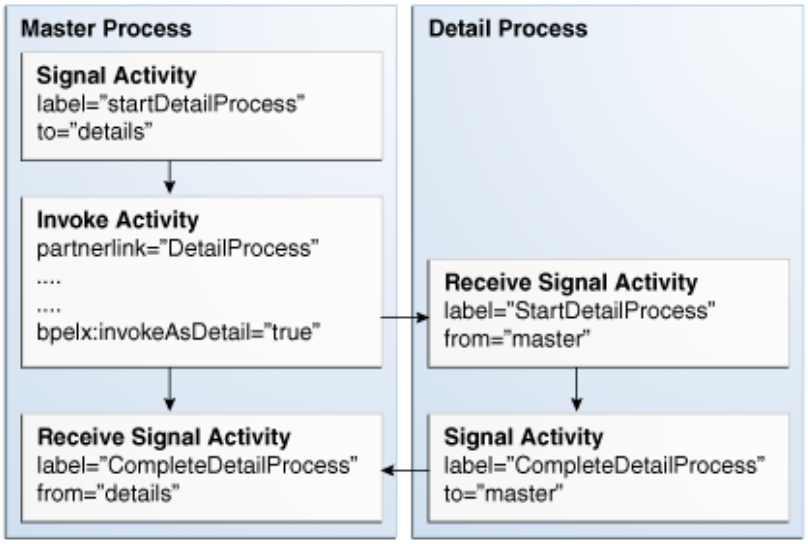
The following BPEL process activities coordinate actions between the master and detail processes:

- 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 16-2 provides an overview of the BPEL process flow coordination.

Figure 16-2 Master and Detail Syntax Overview (One BPEL Process to One Detail Process)



As shown in Figure 16-2, each master and detail process includes a signal and receive signal activity. Table 16-1 describes activity responsibilities based on the type of process in which they are defined.

Table 16-1 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.

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. The following provides an example:

```
<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 in the following example shows a signal activity in a BPEL process that supports BPEL version 2.0.

```
<extensionActivity>
   <bpelx:signal name="notifyDetailProcess"
                 label="startDetailProcess" to="details"/>
</extensionActivity>
```

Note:

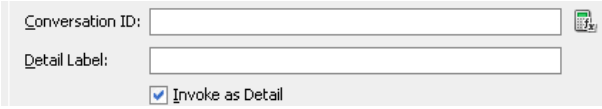
In BPEL 1.1, the signal activity syntax is slightly different.

```
<bpelx:signal name="notifyDetailProcess" label="startDetailProcess"
to="details"/>
```

Assign, invoke, and receive activities describe the interaction between the master and detail processes. This example shows interaction between the master process and one of the detail processes (`DetailProcess`). Similar interaction is defined in this BPEL file for all detail processes.

In the invoke activity, ensure that the **Invoke as Detail** check box is selected. Figure 16-3 provides details.

Figure 16-3 Invoke As Detail Check Box



This selection creates the partner process instance (`DetailProcess`) as a detail instance. You must select this check box in the invoke activity of the master process for each detail process with which to interact. The following provides an example of the BPEL file contents after you select the **Invoke as Detail** check box:

```
<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. The following code provides an example. This syntax shows a receive signal activity in a BPEL process that supports BPEL version 2.0.

```
<extensionActivity>
   <bpelx:receiveSignal name="waitForNotifyFromDetailProcess"
   label="detailProcessComplete" from="details"/>
</extensionActivity>
```

Note:

In BPEL 1.1, the receive signal activity syntax is slightly different.

```
<bpelx:receiveSignal name="waitForNotifyFromDetailProcess"
                     label="detailProcessComplete"
                     from="details"/>
```

Correlating a Master Process with Multiple Detail Processes

For environments in which you have one master and multiple detail processes, use the `bpelx:detailLabel` attribute for signal correlation. The following example shows how to use this attribute.

The first invoke activity invokes the `DetailProcess` detail process and associates it with a label of `detailProcessComplete0`.

```
<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 `DetailProcess1` detail process and associates it with a label of `detailProcessComplete1`. The following provides an example.

```
<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 `DetailProcess2` detail process again through a different port and with a different input variable. It associates the `DetailProcess2` detail process with a label of `detailProcessComplete1-2`, as shown in the following example:

```
<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 the following example waits for a return signal from detail process `DetailProcess0`.

```
<!-- 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 the following example also waits for a return signal from `DetailProcess1` and `DetailProcess2`.

```
<!-- 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.

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 the following example 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.

```
<bpelx:receiveSignal name="waitForNotifyFromMasterProcess"
   label="startDetailProcess" from="master"/>
```

A signal activity indicates that the detail process shown in the following example 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.

```
<bpelx:signal name="notifyMAsterProcess" label="detailProcessComplete"
   to="master"/>
```

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.

How to Create a Master Process

To create a master process:

1. In the , create a BPEL process service component. For this example, the process is named **MasterProcess**.
2. Double-click the **MasterProcess** BPEL process.
3. In the Components window, expand **Oracle Extensions** > **Signal**.
4. Drag a **Signal** activity into the designer.
5. Click the **Signal** activity to display its property fields in the Property Inspector or double-click the **Signal** activity.

   For information about editing activities in the Property Inspector, see How to Edit BPEL Activities in the Property Inspector.

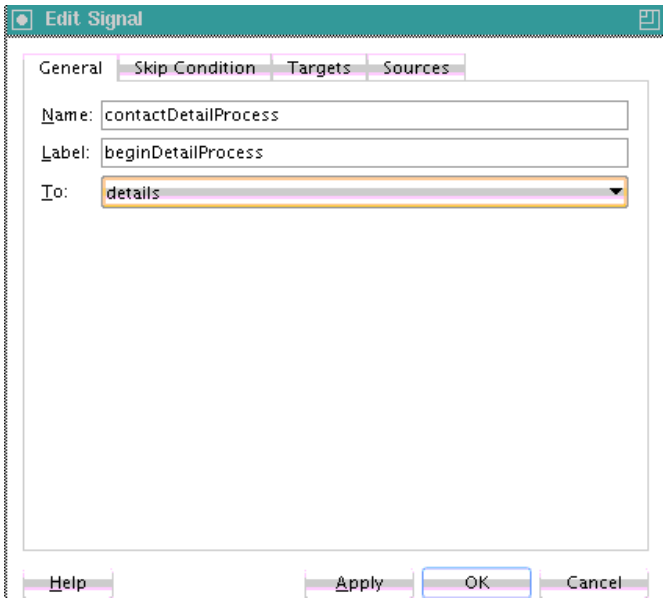
   This activity signals the detail process to perform processing at runtime.
6. Enter the details described in Table 16-2:

Table 16-2 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.
To	Select **details** as the type of process to receive this signal.

Figure 16-4 shows the Signal dialog.

Figure 16-4 Signal Dialog



7. Click **OK**.
8. Drag a **Receive Signal** activity into the designer.
9. Double-click the **Receive Signal** activity.

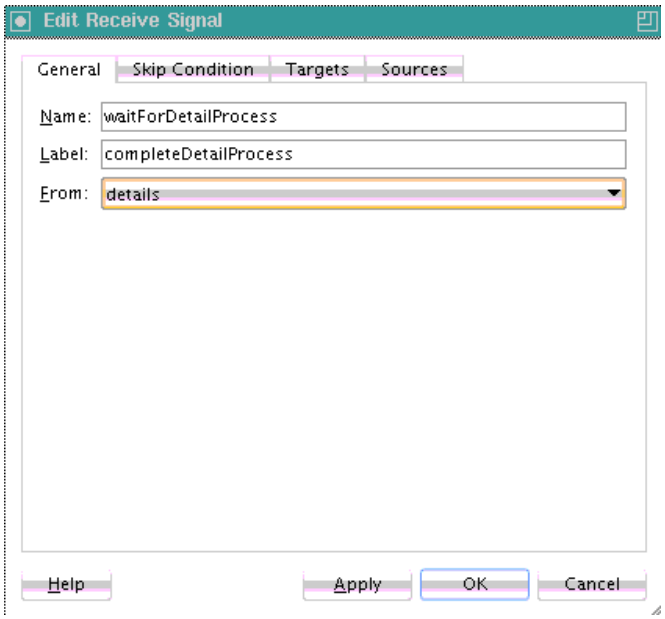
   This activity enables the master process to wait until it receives the signal executed by all of its detail processes.
10. Enter the details shown in Table 16-3:

Table 16-3 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.
To	Select **details** as the type of process from which to receive the signal.

Figure 16-5 shows the Receive Signal dialog.

Figure 16-5 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.

How to Create a Detail Process

To create a detail process:

1. In the , 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 Components window, 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 16-4:

Table 16-4 Receive Signal Dialog Fields and Values

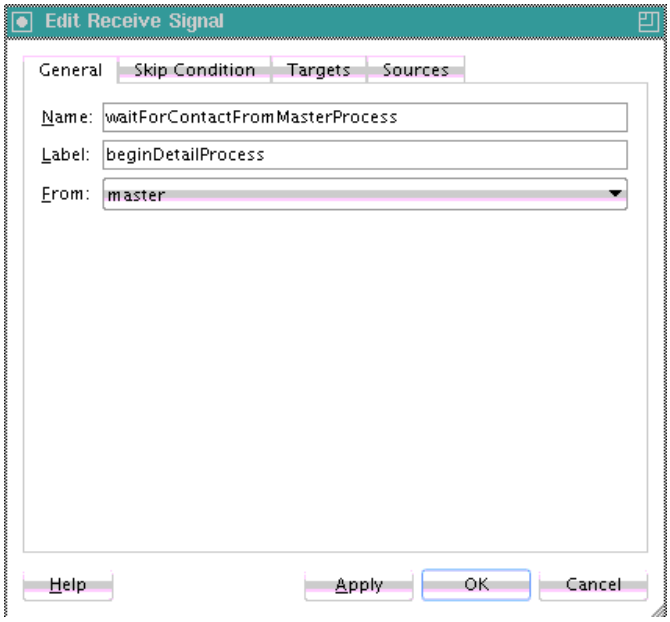
Field	Value
Name	Enter a name (for this example, `WaitForContactFromMasterProcess`).

Table 16-4 (Cont.) Receive Signal Dialog Fields and Values

Field	Value
Label	Enter a label name (for this example, `beginDetailProcess`). This label must match the signal activity label you set in the master process in Step 6.
To	Select **master** as the type of process from which to receive the signal.

Figure 16-6 shows the Receive Signal dialog.

Figure 16-6 Receive Signal Dialog



7. Click **OK**.
8. Drag a **Signal** activity into the designer.
9. Double-click the **Signal** activity.

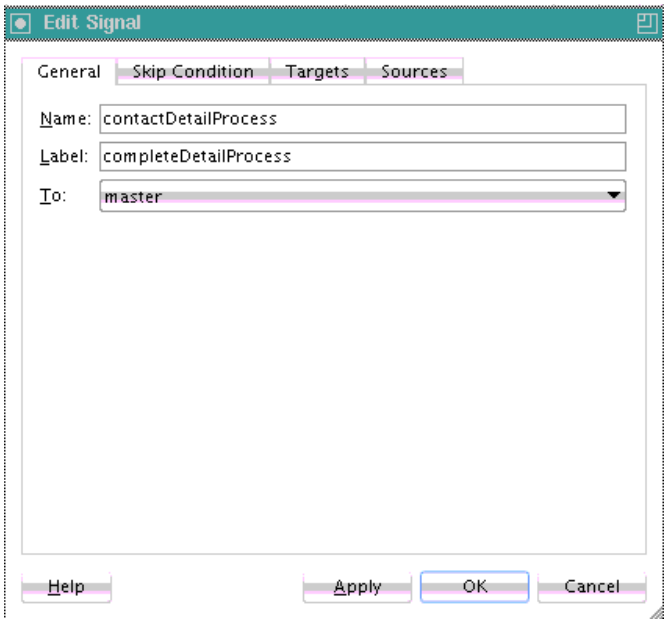
   This activity enables the detail process to signal its associated master process at runtime that processing is complete.
10. Enter the details described in Table 16-5:

Table 16-5 Signal Dialog Fields and Values

Field	Value
Name	Enter a name (for this example, `contactDetailProcess`).
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.
To	Select **master** as the destination.

Figure 16-7 shows the Signal dialog.

Figure 16-7 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.

How to Create an Invoke Activity

To create an invoke activity:

1. Return to the **MasterProcess** master process.
2. In the Components window, 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 as the partner link.
6. Select the **Invoke as Detail** check box.
7. Complete all remaining fields in the Invoke dialog, and click **OK**.
8. In the designer, click **Source**. The BPEL file appears as follows:

```
<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:

   ```
   <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:

   ```
   <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.

17

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:

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. You 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

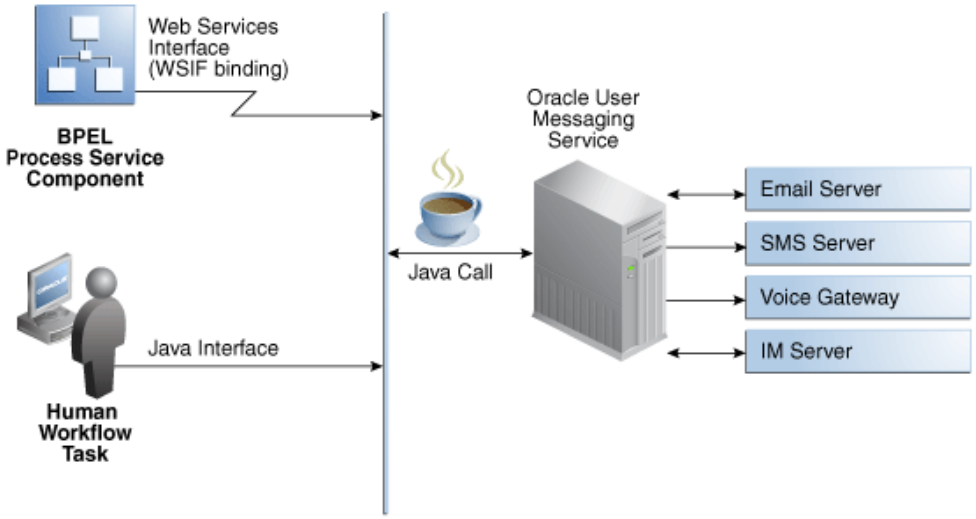
  The BPEL notification service uses the underlying infrastructure provided by Oracle User Messaging Service to send notifications.

  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 *Developing Applications with Oracle User Messaging Service*.

  Figure 17-1 shows the Oracle User Messaging Service interfaces and supported service types.

Figure 17-1 Service Interfaces and Supported Service Types



For more information about notifications, see the following sections:

- Notifications from Human Workflow
- Specifying Participant Notification Preferences for instructions on specifying email notifications in the Human Task Editor
- *Developing Applications with Oracle User Messaging Service*

Introduction to Notification Channel Setup

Notification setup is a multiple-step process that involves several user interface tools. Table 17-1 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 17-1 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	Selecting Notification Channels During BPEL Process Design or 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	*Administering Oracle SOA Suite and Oracle Business Process Management 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	*Administering Oracle SOA Suite and Oracle Business Process Management 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. 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.	*Developing Applications with Oracle User Messaging Service*

Selecting Notification Channels During BPEL Process Design

Oracle JDeveloper includes the email, IM, SMS, and voice channel notification channels in the Components window. 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 Components window, expand **Oracle Extensions**.
2. Go to the **Notification** section.
3. Drag a notification channel into the designer:
   - **Email**
   - **IM**
   - **SMS**
   - **Voice**
4. See the section in Table 17-2 based on the notification channel you selected.

Table 17-2 Notification Channels

If You Selected...	See...
Email	How To Configure the Email Notification Channel to configure email notification
IM	How to Configure the IM Notification Channel to configure IM notification
SMS	How to Configure the SMS Notification Channel to configure SMS notification
Voice	How to Configure the Voice Notification Channel to configure voice message notification

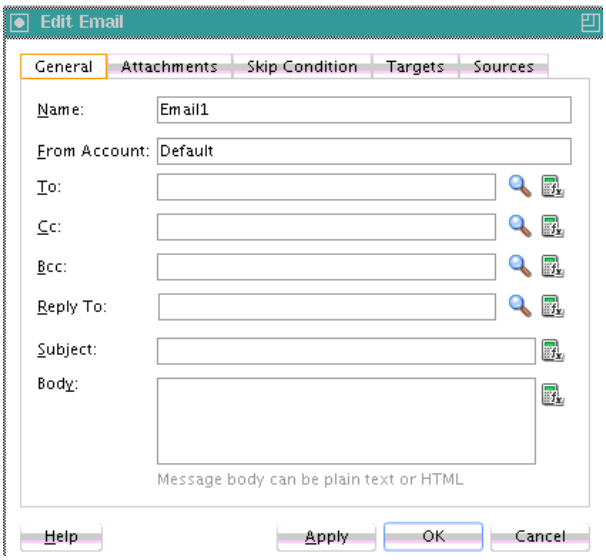
Note:

If you delete an email, voice, SMS, or IM activity, any partner link with which it is integrated is not automatically deleted.

How To Configure the Email Notification Channel

When you drag the **Email** icon from the Components window, the Email dialog appears. Figure 17-2 shows the required email notification parameters.

Figure 17-2 Email Dialog



To configure the email notification channel:

1. Enter information for each field as described in Table 17-3.

Note:

For the **To**, **CC**, and **Bcc** fields, separate multiple addresses with a semicolon (;).

Table 17-3 Email Notification Parameters

Name	Description
Name	Enter a name or accept the default name of `Email`*`Number`*.
From Account	The name of the account used to send this message. The default account is named **Default** and is editable from the **Mailer** tab of 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 *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

Table 17-3 (Cont.) Email Notification Parameters

Name	Description
To	The email address to which to deliver the message. 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 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 also be a static or dynamic address, as described for the **To** address.
Reply To	The email address to use for replies. This can also be a static or dynamic address, as described for the **To** address.
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 also be plain text, HTML, or dynamic text, as described for the **Subject** parameter.

2. Click **OK**.

   The BPEL fragment that invokes the notification service to send the email message is created.

3. See Table 17-1 of Introduction to Notification Channel Setup for additional configuration procedures to perform outside of Oracle JDeveloper.

   The following example uses an email activity in a scope named **Scope_NotifyCustomerofCompletion**. 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:

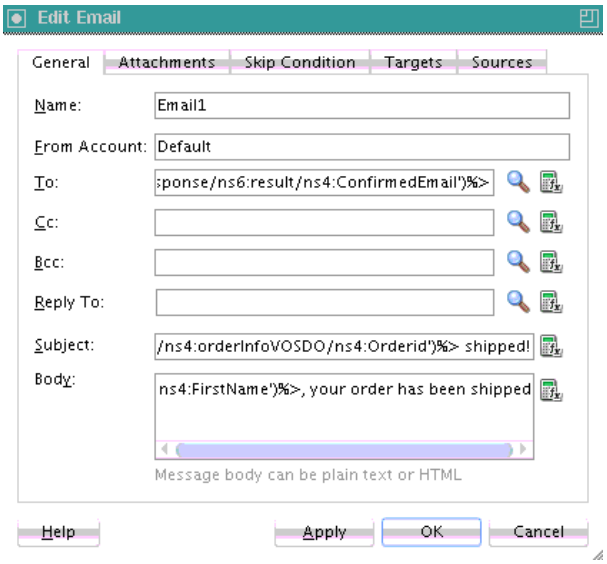
     ```
     Order with id
     <%bpws:getVariableData('gOrderInfoVariable','/ns2:orderInfoVOSDO/ns2:OrderI
     d')%> shipped!
     ```

   - 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 17-3 provides details.

Figure 17-3 Email Dialog



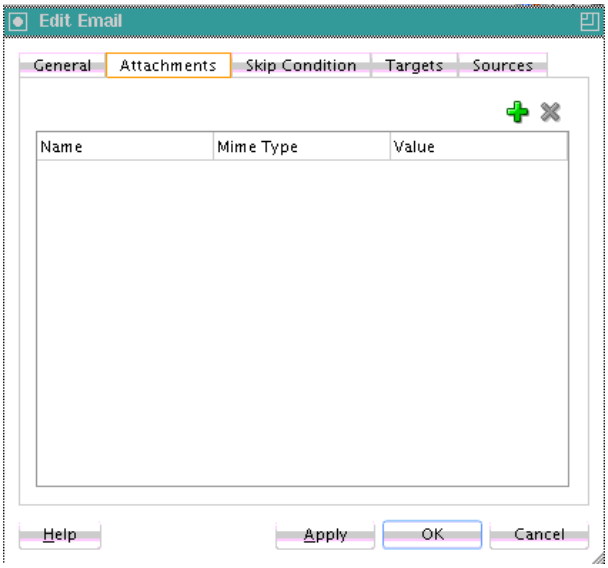
Setting Email Attachments

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 Components window, select **Email** as the notification channel.
2. Specify values for **To**, **Subject**, and **Body**.
3. Click the **Attachments** tab. Figure 17-4 provides details.

Figure 17-4 Attachments Tab



4. Click the **Add** icon to add as many attachments as you require. The number of attachments does not need to include the body part.

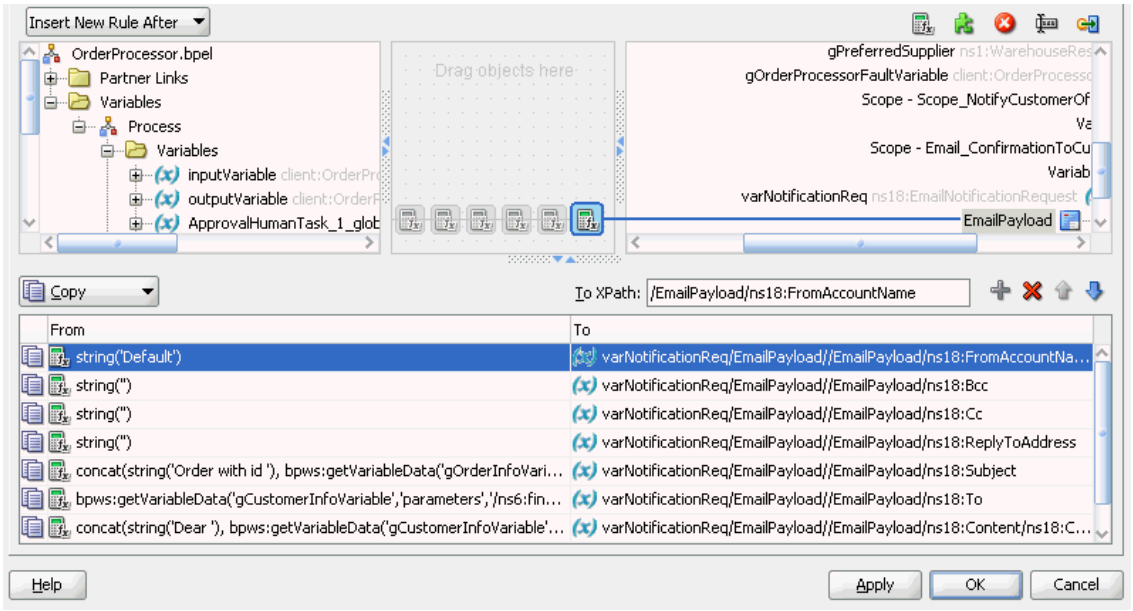
5. In the **Name** field, change the name or accept the default value of `Attachment`*`number`*.
6. In the **Mime Type** field, click the **Browse** icon to invoke the Expression Builder dialog for adding MIME type contents.
7. When complete, click **OK** to return to the **Attachments** tab.
8. In the **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 in Oracle BPEL Designer.

    An assign activity named **EmailParamsAssign** appears.
12. Double-click **EmailParamsAssign**.

    Note the settings in **EmailParamsAssign**, as shown in Figure 17-5.

Figure 17-5 EmailParamsAssign Assign Activity



For more information about sending attachments using email, see the following documentation:

- *Developing Applications with Oracle User Messaging Service*
- *Administering Oracle User Messaging Service*

Formatting the Body of an Email Message as HTML

You can format the body of an email message as HTML instead of 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 the following example:

```
<copy>
 <from
expression="ora:processXSLT('TransformPositionSummary7.xslt',bpws:
getVariableData('ClientPositionSummary'))"/>
 <to variable="varNotificationReq" part="EmailPayload"
query="/EmailPayload/ns9:Content/ns9:ContentBody"/>
</copy>
```

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 the following example for the message content inside the 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>
```

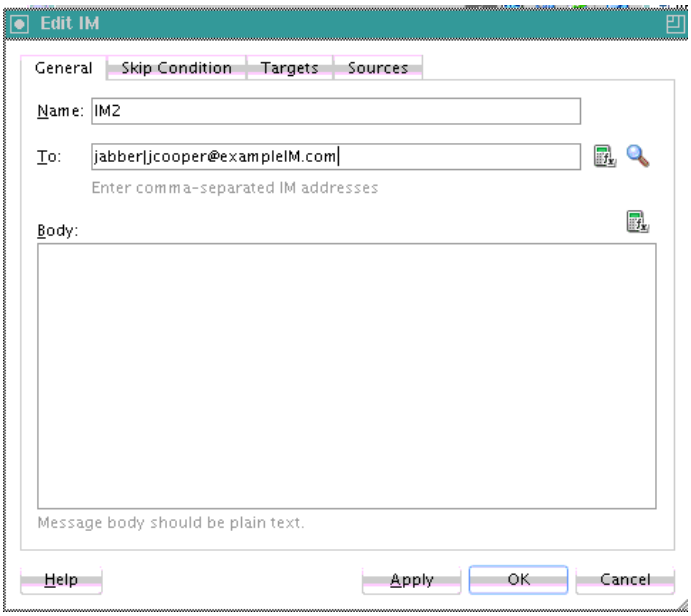
For this to work correctly, you must pass the output of the `processXSLT()` function to the `CDATA()` function, as shown in the following example:

```
<%ora:toCDATA(xdk:processXSLT('xsl/email.xslt',
 bpws:getVariableData('inputVariable','payload','/client:process/client:input')
 ))%>
```

How to Configure the IM Notification Channel

When you drag the **IM** icon from the Components window, the IM dialog appears. Figure 17-6 shows the required IM notification parameters.

Figure 17-6 IM Dialog



To configure the IM notification channel:

1. Enter information for each field as described in Table 17-4.

Table 17-4 IM Notification Parameters

Name	Description
Name	Enter a name or accept the default name of `IM`*`Number`*.
To	The IM address to which to deliver the message. 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**.

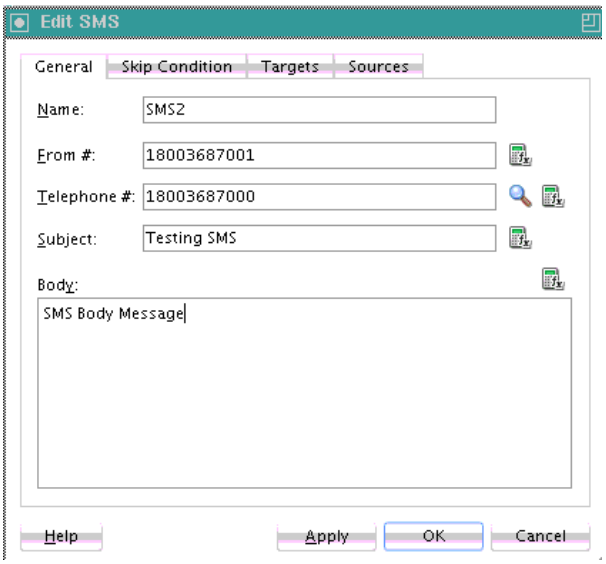
   The BPEL fragment that invokes the notification service for IM notification is created.

3. See Table 17-1 of Introduction to Notification Channel Setup for additional configuration procedures to perform outside of Oracle JDeveloper.

How to Configure the SMS Notification Channel

When you drag the **SMS** icon from the Components window, the SMS dialog appears. Figure 17-7 shows the required SMS notification parameters.

Figure 17-7 SMS Dialog



To configure the SMS notification channel:

1. Enter information for each field as described in Table 17-5.

Table 17-5 SMS Notification Parameters

Name	Description
Name	Enter a name or accept the default name of `SMS`*`Number`*.
From #	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 How to Select Email Addresses and Telephone Numbers Dynamically.
Telephone #	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**.

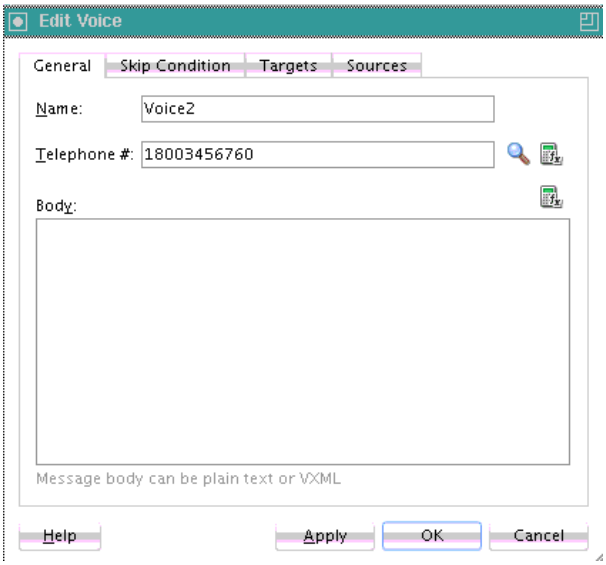
The BPEL fragment that invokes the notification service for SMS notification is created.

3. See Table 17-1 of Introduction to Notification Channel Setup for additional configuration procedures to perform outside of Oracle JDeveloper.

How to Configure the Voice Notification Channel

When you drag the **Voice** icon from the Components window, the Voice dialog appears. Figure 17-8 shows the required voice notification parameters.

Figure 17-8 Voice Dialog



To configure the voice notification channel:

1. Enter information for each field as described in Table 17-6.

Table 17-6 Voice Notification Parameters

Name	Description
Name	Enter a name or accept the default name of `Voice`*`Number`*.
Telephone #	The telephone number to which to deliver the message. 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 17-1 of Introduction to Notification Channel Setup for additional configuration procedures to perform outside of Oracle JDeveloper.

How to Select Email Addresses and Telephone Numbers Dynamically

You can 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 17-7 lists the property names that can be used with this XPath function.

Table 17-7 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');
```

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 #** field on the SMS and Voice dialogs

For more information about using the Identity Lookup dialog, see Introduction to Human Workflow Services.

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 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 How to Send Task Attachments with Email Notifications.

For more information about the Oracle User Messaging Service, see the "User Communication Preferences" chapter of *Developing Applications with Oracle User Messaging Service*.

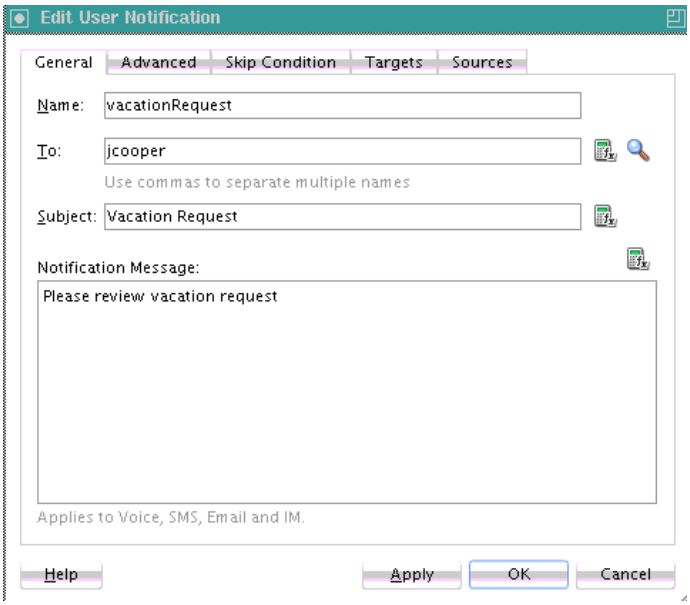
For information about configuring the Oracle User Messaging Service in Oracle Enterprise Manager Fusion Middleware Control, see *Administering Oracle User Messaging Service*.

How to Allow the End User to Select Notification Channels

To allow the end user to select notification channels:

1. From the Components window list, expand **Oracle Extensions**.
2. From the **Notification** section, drag the **User Notification** activity into the designer. Figure 17-9 shows the required user notification parameters.

Figure 17-9 User Notification Dialog



3. Enter information for each field as described in Table 17-8.

Table 17-8 User Notification Parameters

Name	Description
Name	Enter a name or accept the default name of `UserNotification`*`Number`*.
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.

Table 17-8 (Cont.) User Notification Parameters

Name	Description
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.

4. Click **Apply**.

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:

- Oracle BPEL Designer specifies the users named `jcooper` and `jstein` in the **General** tab.
- Oracle 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"
```

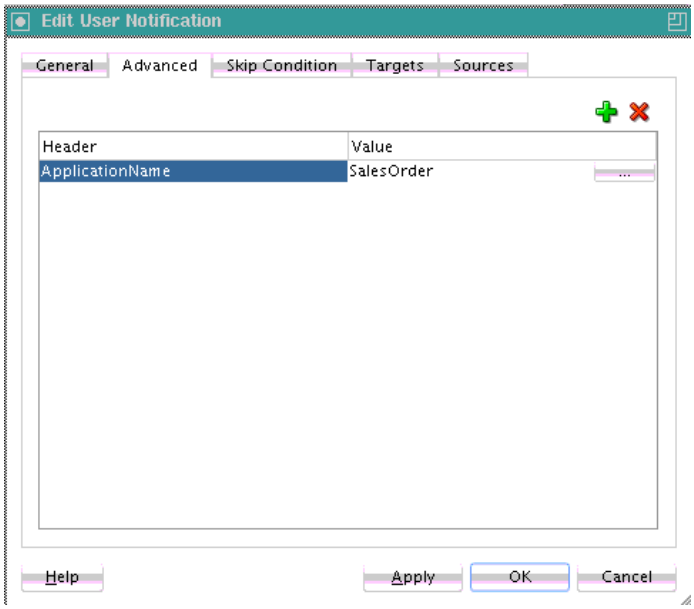
- The end user `jstein` creates the following preference rule in the User Messaging Preferences user interface:

```
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 17-10 shows the **Advanced** tab of the User Notification dialog.

Figure 17-10 User Notification Advanced Parameters



2. Click the **Add** icon to add a row to the **Header** and **Name** columns.
3. In the **Header** column, click the field to display a list for selecting a value. Otherwise, manually enter a value.
4. In the **Name** column, enter a value.
5. Click **OK**.

18

Using Oracle BPEL Process Manager Sensors and Analytics

This chapter describes how to use sensors to select BPEL activities, variables, and faults to monitor during runtime in a BPEL process. It also describes how to create sensor actions to publish the values of sensors to an endpoint.

This chapter includes the following sections:

- Introduction to Sensors
- Configuring Sensors and Sensor Actions in Oracle JDeveloper
- Viewing Sensors and Sensor Action Definitions in Oracle Enterprise Manager Fusion Middleware Control
- Configuring BPEL Process Analytics

For more information about Oracle BPEL Process Manager sensors, see Understanding Sensor Public Views and the Sensor Actions XSD .

Introduction to Oracle BPEL Process Manager 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 following:

- The value of the variable at the moment it was modified
- A time stamp when the variable was modified
- 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 BAM, the sensor value can be sent to the Oracle 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 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 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

For information about how these files are created, see How to Configure Activity_ Variable_ and Fault Sensors and How to Configure Sensor Actions.

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 following directory:

    `/soa/integration/seed/soa/shared/bpel/Sensor.xsd`

    The `Sensor.xsd` file is also included in the following directory:

    `/soa/integration/jdeveloper/seed/soa/shared/bpel/Sensor.xsd`

  - 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.

Composite Sensors

While BPEL sensors are used to declare interest in specific events throughout the life cycle of a BPEL process instance, 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.
- Publish JMS data computed from incoming and outgoing messages.
- Track composite instances initiated through business event subscriptions.

For information about composite sensors, see Defining Composite Sensors .

Configuring Sensors and Sensor Actions in Oracle JDeveloper

In Oracle JDeveloper, sensor actions and sensors are displayed as part of **Monitor** view.

How to Access Sensors and Sensor Actions

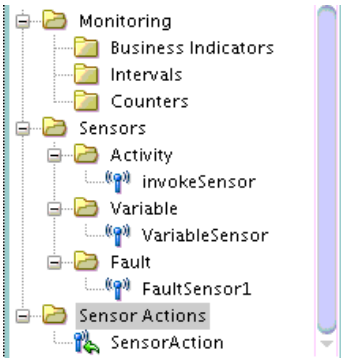
To access sensors and sensor actions:

1. Select **Change to Monitor view** at the top of Oracle BPEL Designer, as shown in Figure 18-1.

Figure 18-1 Monitor View

Figure 18-2 shows the sensor actions and sensors in the Structure window.

Figure 18-2 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** folder and selecting **Create** > **Sensor Action.**

3. Add activity sensors, variable sensors, or fault sensors as follows:

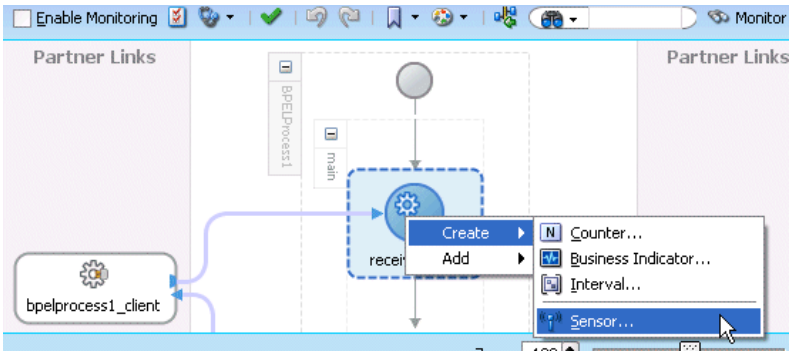
   a. Expand the **Sensors** folder.

   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 18-3 provides details.

Figure 18-3 Creating an Activity Sensor



The following sections describe how to configure sensors and sensor actions.

How to Configure Activity, Variable, and Fault Sensors

This section describes how to configure activity, variable, and fault sensors.

To Configure an Activity Sensor:

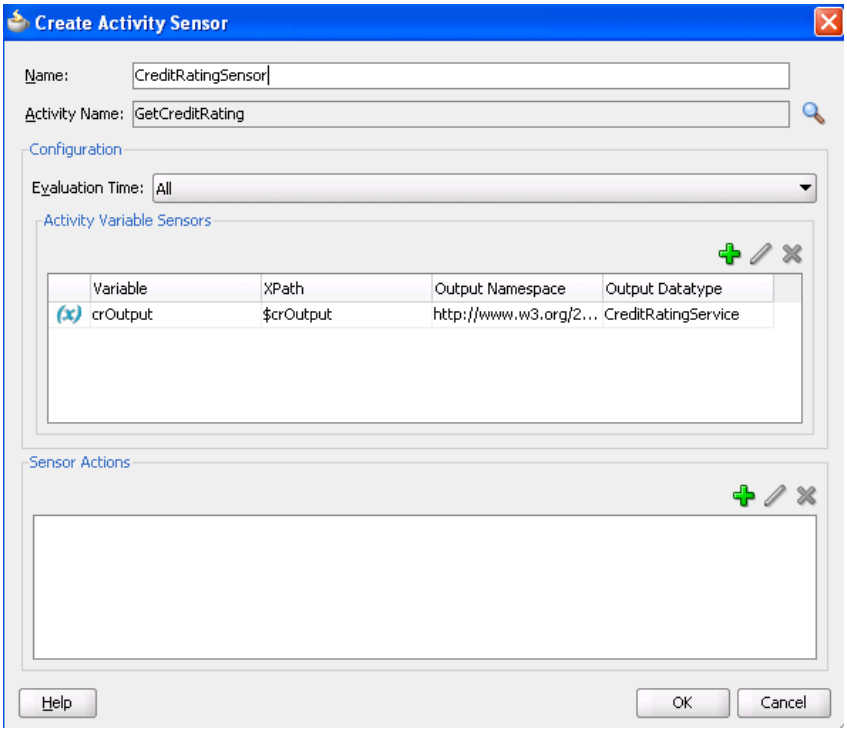
Assume you are monitoring a loan flow application, and want to know the following:

- 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 18-4.

1. Select **Change to Monitor view** 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 18-4 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 18-4 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:

```
<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.
7. If you want to add a sensor action on the activity, then in the **Sensor Actions** section, click the **Add** icon. For more information, see How to Configure Sensor Actions.
8. Click **OK**.

Note:

If you did not specify any values in the **Activity Variable Sensors** and **Sensor Actions** sections, you do not receive any validation errors or warning messages in the Log window in Oracle JDeveloper or in any log files. This is the expected behavior.

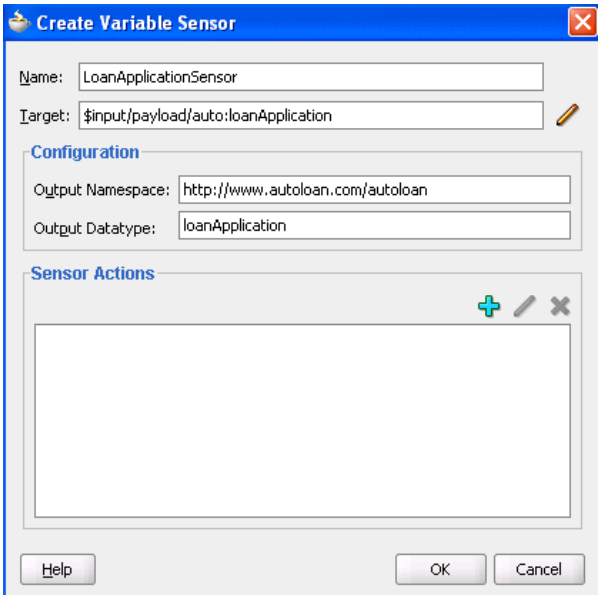
To Configure a Variable Sensor:

If you want to record all incoming loan requests, you can create a variable sensor.

1. Select **Change to Monitor view** at the top of Oracle BPEL Designer.

2. In the Structure window, expand the **Sensors** folder.
3. Right-click **Variable**, and select **Create**.
4. 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 18-5.

Figure 18-5 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:

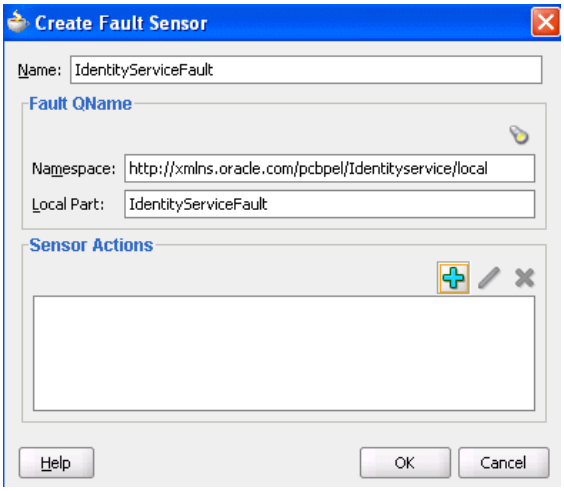
```
<sensor sensorName="LoanApplicationSensor"
    classname="oracle.tip.pc.services.reports.dca.agents.BpelVariableSensorAgent"
    kind="variable"
    target="$input/payload">
  <variableConfig outputNamespace="http://www.autoloan.com/ns/autoloan"
                  outputDataType="loanApplication"/>
</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. Select **Change to Monitor view** at the top of Oracle BPEL Designer.
2. In the Structure window, expand the **Sensors** folder.
3. Right-click **Fault**, and select **Create**.
4. Click the **Browse** icon above the **Namespace** field to select to create a fault sensor, as shown in Figure 18-6.

Figure 18-6 Creating a Fault Sensor



Based on your selection, the **Namespace** and **Local Parts** fields are automatically filled in.

5. If you want to add a sensor action on the fault, then in the **Sensor Actions** section, click the **Add** icon. For more information, see How to Configure Sensor Actions.

6. Click **OK**.

A new entry is created in the *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>
```

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 18-5 to a JMS queue), then create a sensor action, as shown in Figure 18-7, and associate it with the **LoanApplicationSensor** variable.

To configure a sensor action:

1. Select **Change to Monitor view** at the top of Oracle BPEL Designer.

2. In the Structure window, right-click the **Sensor Actions** folder.

3. Select **Create** > **Sensor Action**.

4. Enter the details described in Table 18-1.

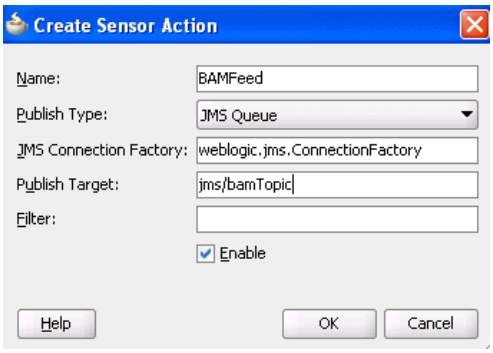
Table 18-1 Sensor Actions Dialog

Field	Description
Name	Enter a name or accept the default name.

Table 18-1 (Cont.) Sensor Actions Dialog

Field	Description
Publish Type	Select the destination to which to publish sensor data. For more information, see section Introduction to Sensors.
JMS Connection Factory	If your publish type is **JMS Queue**, **JMS Topic**, or **JMS Adapter**, specify the connection factory.
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 18-9.
Enable	Deselect this check box to disable a sensor action. By default, sensor actions are enabled. If you disable a sensor action by deselecting this check box, the action does not publish data.

Figure 18-7 Creating a Sensor Action



A new entry is created in the *bpel_process_name*_sensorAction.xml file:

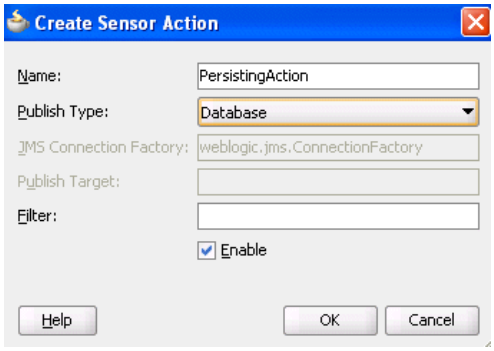
```
<action name="BAMFeed"
        enabled="true"
        publishType="JMSQueue"
        publishTarget="jms/bamTopic">
  <sensorName>LoanApplicationSensor</sensorName>
  <property name="JMSConnectionFactory">
    weblogic.jms.ConnectionFactory
  </property>
</action>
```

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.

5. 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 18-8, and associate it with both **CreditRatingSensor** and **LoanApplicationSensor**.

Figure 18-8 Creating an Additional Sensor Action



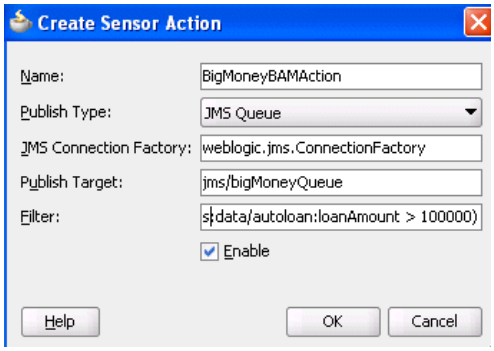
A new entry is created in the *`bpel_process_name`*`_sensorAction.xml` file:

```
<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.

6. If you want to monitor loan requests for which the loan amount is greater than $100,000, create a sensor action with a filter, as shown in Figure 18-9. There is no design-time validation of the filter query. You must ensure the query is correct.

Figure 18-9 Creating a Sensor Action with a Filter



A new entry is created in the *bpel_process_name*_sensorAction.xml file:

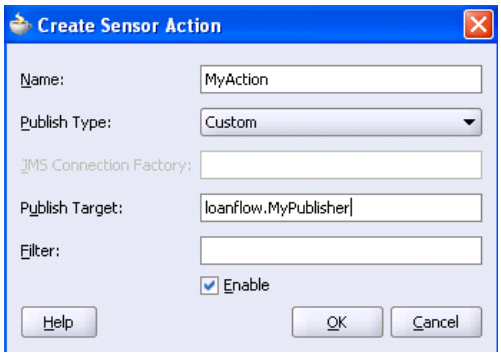
```
<action name="BigMoneyBAMAction"
        enabled='true'
        filter="boolean(/s:actionData/s:payload
                        /s:variableData/s:data
                        /autoloan:loanAmount > 100000)"
        publishType="JMSQueue"
        publishTarget="jms/bigMoneyQueue">
  <sensorName>LoanApplicationSensor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

Note:

- 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.

7. 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 18-10. The name in the **Publish Target** field denotes a fully qualified Java class name that must be implemented. For more information, see How to Create a Custom Data Publisher.

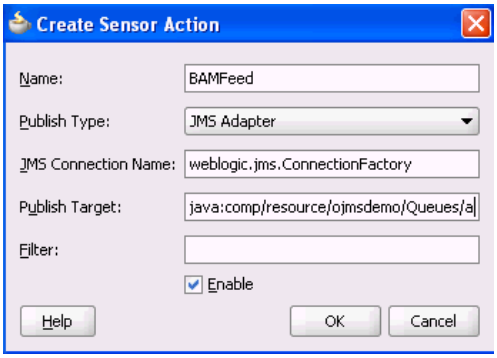
Figure 18-10 Using the Custom Publish Type



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 18-11.

Figure 18-11 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 . 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 *Understanding Technology Adapters*.

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 Applications window, double-click the BPEL project.

   The Project Properties dialog appears.

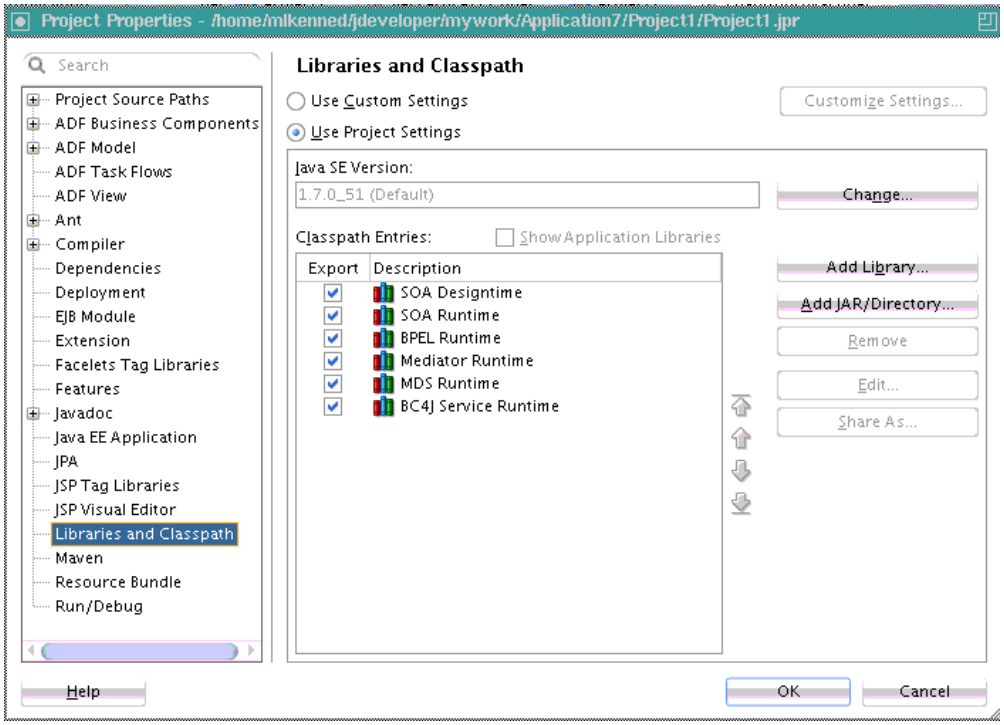
2. Click **Libraries and Classpath**.

3. Browse and select the following:

   SOA_ORACLE_HOME`/lib/java/shared/oracle.soainfra.common/11.1.1/orabpel.jar`

   Figure 18-12 provides details.

Figure 18-12 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 Oracle JDeveloper, implement the publish method of the **DataPublisher** interface, as shown in the sample custom data publisher class in Figure 18-13.

Figure 18-13 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.

How to Register the Sensors and Sensor Actions in the composite.xml File

Oracle JDeveloper automatically updates the `composite.xml` file to include appropriate properties for sensors and sensor actions, as shown in the following exxample:

```
<composite name="JMSQFComposite" applicationName="JMSQueueFilterApp"
   revision="1.0" label="2007-04-02_14-41-31_553" mode="active" state="on">
  <import namespace="http://xmlns.oracle.com/JMSQueueFilter"
  location="JMSQueueFilter.wsdl" importType="wsdl"/>
    <service name="client">
      <interface.wsdl interface="http://xmlns.oracle.com/
         JMSQueueFilter#wsdl.interface(JMSQueueFilter)"/>
```

```
      <binding.ws
      port="http://xmlns.oracle.com/JMSQueueFilter#wsdl.endpoint(client/
         JMSQueueFilter_pt)"/>
    </service>
    <component name="JMSQueueFilter">
    <implementation.bpel src="JMSQueueFilter.bpel"/>
    <property name="configuration.sensorLocation" type="xs:string"
    many="false">JMSQueueFilter_sensor.xml</property>
    <property name="configuration.sensorActionLocation" type="xs:string"
    many="false">JMSQueueFilter_sensorAction.xml</property>
</component>
<wire>
   <source.uri>client</source.uri>
   <target.uri>JMSQueueFilter/client</target.uri>
</wire>
</composite>
```

You can specify additional properties with `<property name= ...>`, as shown in the preceding example.

Viewing Sensors and Sensor Action Definitions in Oracle Enterprise Manager Fusion Middleware Control

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 *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

Note:

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.

Configuring BPEL Process Analytics

BPEL process analytics provide the following features:

- A uniform measurement mechanism across Oracle SOA Suite components such as Oracle BPMN, human workflow, and BPEL processes for collecting disparate data.
- A runtime infrastructure for evaluating, publishing, and synthesizing measurement events.

For information about BPEL process analytics integration with Oracle Business Activity Monitoring (BAM), see Chapter "Integrating with Oracle SOA Suite" of *Monitoring Business Activity with Oracle BAM* and Chapter "Gaining Business Insights with Oracle Business Activity Monitoring" of *Understanding Oracle SOA Suite*.

Introduction to Business Indicators

Business indicators are defined in a SOA composite application to identify objects that contribute to the analytical and metric calculations of components. Business indicators consist of the following types:

- Measures

  Store the values of a variable such as a sales amount, an employee salary, and so on. Measures only enable data types that are continuous, and are typically numeric values.

- Dimensions

  Label group or filter measures.

- Counters

  Track the number of times a process instance completes a marked element.

Metadata specified dimensions and measures are captured as part of the measurement.

Business indicators are designed to be sharable and bindable to multiple BPEL processes within the composite. This enables you to monitor their value changes from one process to another when the composite is executed during analytics runtime.

Introduction to Standard Sampling Points

Standard sampling points are points in a component path at which the component inherently attempts to create a measurement event. Measurement metadata can configure measurements at these standard sampling points. If appropriate measurement metadata exists that enables some or all of the standard sampling point measurement events, then these measurement events are generated, published, and processed. For example, a standard sampling point in a process can be the following:

- Start and stop a process
- Start and stop an activity
- Faults

Introduction to User-Defined Sampling Points

These are the sampling points that you can specify on a component:

- Measurement mark:

  A single point of measurement for the specified measure.

- Measurement interval:

  A measurement consisting of a starting point and ending point (therefore, constituting an interval identified by a measurement interval name) typically along the path taken by a component.

- Measurement counter:

  A measurement that identifies the occurrence of a specific point in the path taken by a component.

Measurements are a combination of a sampling point and a selected business indicator executed at runtime. For more information about measurements, see How to Define Measurements.

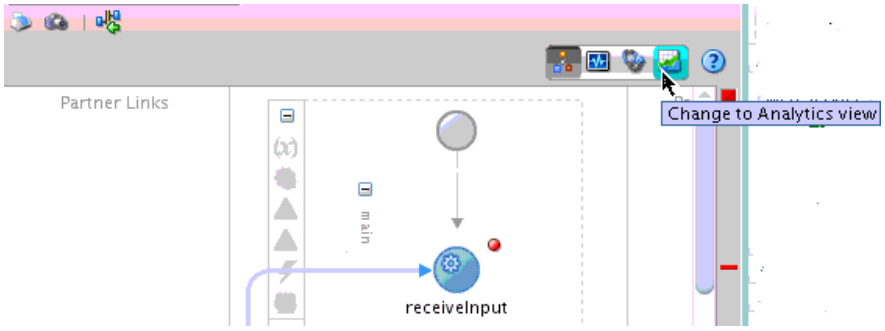
How to Access Analytics View

You edit business indicators and measurements in analytics view of a BPEL process in Oracle BPEL Designer.

To access analytics view:

1. In the SOA Composite Editor, double-click a BPEL process.
2. Above the BPEL process in Oracle BPEL Designer, click **Change to Analytics view**. Figure 18-14 provides details.

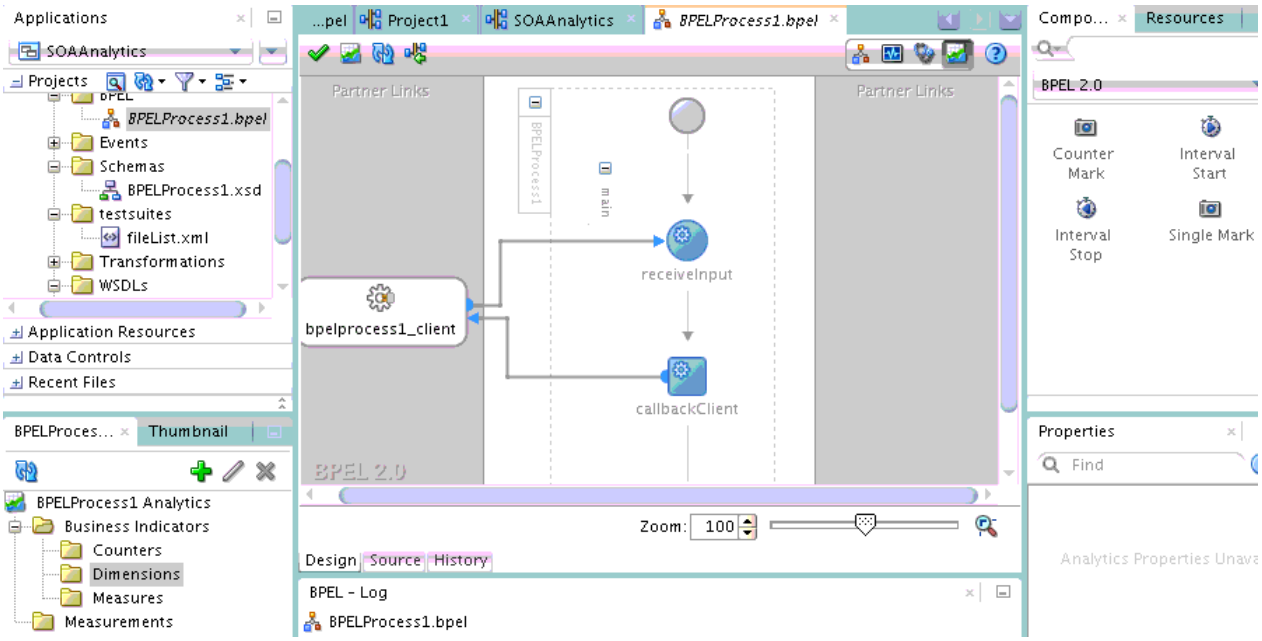
Figure 18-14 Analytics View Icon in Oracle BPEL Designer



This displays the BPEL process in analytics view, as shown in Figure 18-15.

- The Components window displays a palette of measurement marks and intervals that can be dragged onto BPEL process activities.
- The Structure window displays business indicators for creating counters, dimensions, and measures.

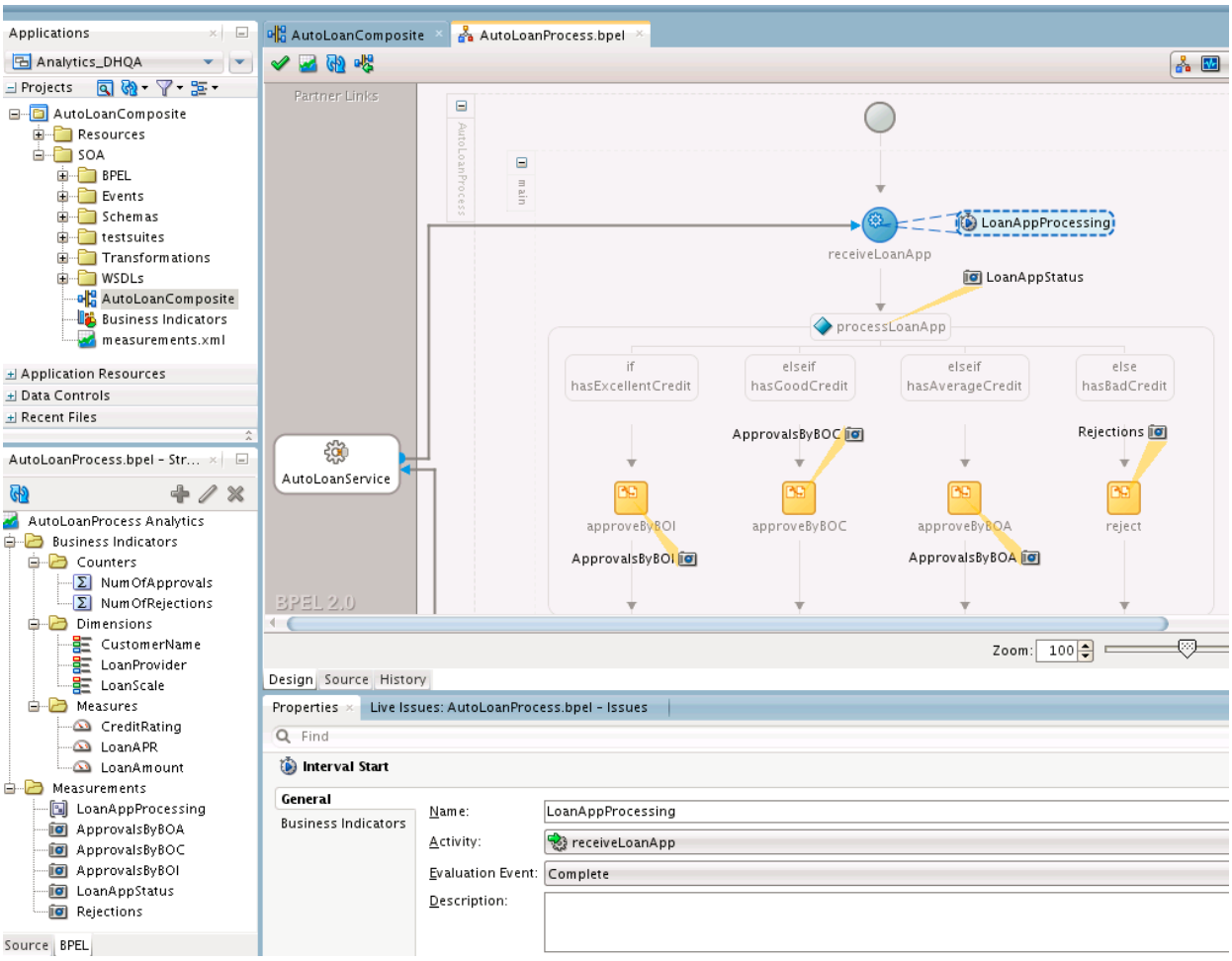
Figure 18-15 Analytics View of a BPEL Process



When business indicator and measurement design is complete, analytics view looks similar to that shown in Figure 18-16.

- Business indicators (counters, dimensions, and measures) and measurements (intervals and marks) defined for the BPEL process are displayed in the Structure window. You can create, edit, and delete business indicators from the Structure window. You can edit and delete, but not create, measurements from the Structure window. Measurements are created by dragging the appropriate icon from the Components window.
- Measurement intervals and marks are defined as floaters on top of the read-only activities in the BPEL process. The measurement floaters can be moved around by mouse on top of activities in the BPEL process to achieve the necessary topology.
- The Property Inspector at the bottom of Oracle BPEL Designer enables you to edit the selected business indicator or measurement. Changes are automatically committed.

Figure 18-16 Analytics View with Business Indicator and Measurement Design Complete



How to Define Business Indicators

You can bind business indicators to BPEL XPath expression functions during creation. Business indicators are designed to be sharable and bindable to multiple BPEL processes within the composite. This enables you to monitor their value changes from one process to another when the composite is executed during analytics runtime.

You can define the following business indicators in a BPEL process:

- Define a counter binding for the BPEL process. An available counter is selected and bound to the BPEL process without the need to specify any XPath expression. A counter is meant to count how many times a certain BPEL activity gets executed at runtime. This means there is no need to specify any XPath expression for the binding.
- Define a dimension binding for the BPEL process. An available dimension is selected and bound to a BPEL XPath expression.
- Define a measure binding for the BPEL process. An available measure is selected and bound to a BPEL XPath expression.

For more information about business indicators, see Introduction to Business Indicators.

After definition, you can edit and delete business indicators in the Business Indicator Overview Editor described in How to Edit Business Indicators in the Business Indicator Overview Editor.

Defining Counters

You can define business indicator counters.

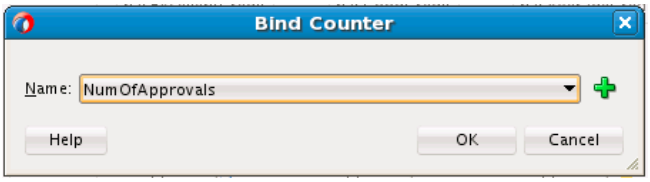
To define counters:

1. Access analytics view in a BPEL process as described in How to Access Analytics View.
2. In the Structure window, right-click **Counters** and select **Create**.

   The Bind Counter dialog is displayed.
3. Select a name, and click **OK**. If there is no counter to which to bind, click the **Add** icon to create a new counter. You can also create counters in the Business Indicator Overview Editor that are then displayed for selection in this dialog. For more information, see How to Edit Business Indicators in the Business Indicator Overview Editor.

   When complete, the Bind Counter dialog looks as shown in Figure 18-17.

Figure 18-17 Bind Counter Dialog



Defining Dimensions

You can define business indicator dimensions.

To define dimensions:

1. Access analytics view in a BPEL process as described in How to Access Analytics View.
2. In the Structure window, right-click **Dimensions** and select **Create**.

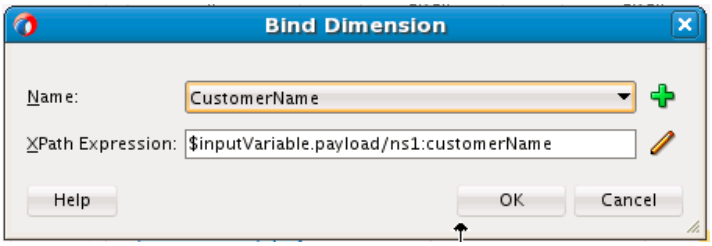
   The Bind Dimension dialog is displayed.
3. Enter values appropriate to your environment, and click **OK**. Table 18-2 provides details.

Table 18-2 Bind Dimension Dialog

Element	Description
Name	Select a name. If there is no dimension to which to bind, click the **Add** icon to invoke the Create Dimension dialog to enter a name and select a data type (boolean, decimal, integer, string or time) for the dimension. You can also create dimensions in the Business Indicator Overview Editor that are then displayed for selection in this dialog. For more information, see How to Edit Business Indicators in the Business Indicator Overview Editor. **Note**: Optional ranges can be specified for some data types such as integers and decimals. This enables the dimensions to show their ranges at analytics runtime for better reporting.
XPath Expression	Click the **Edit** icon to invoke the Expression Builder dialog in which to build an XPath expression for binding to the dimension.

When complete, the Bind Dimension dialog looks as shown in Figure 18-18.

Figure 18-18 Bind Dimension Dialog



Defining Measures

You can define business indicator measures.

To define measures:

1. Access analytics view in a BPEL process as described in How to Access Analytics View.
2. In the Structure window, right-click **Measures** and select **Create**.

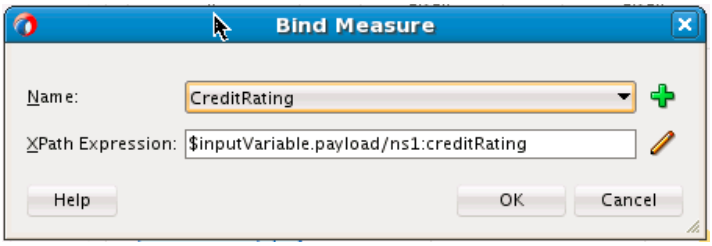
   The Bind Measure dialog is displayed.
3. Enter values appropriate to your environment, and click **OK**. Table 18-3 provides details.

Table 18-3 Bind Measure Dialog

Element	Description
Name	Select a name. If there is no measure to which to bind, click the **Add** icon to invoke the Create Measure dialog to enter a name and select a measure (decimal or integer). You can also create measures in the Business Indicator Overview Editor that are then displayed for selection in this dialog. For more information, see How to Edit Business Indicators in the Business Indicator Overview Editor.
XPath Expression	Click the **Edit** icon to invoke the Expression Builder dialog in which to build the XPath expression for binding to the measure.

When complete, the Bind Measure dialog looks as shown in Figure 18-19.

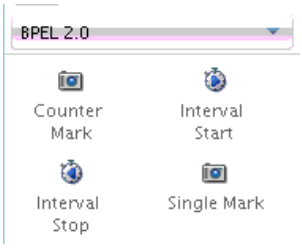
Figure 18-19 Bind Measure Dialog



How to Define Measurements

The Components window consists of the measurement types shown in Figure 18-20:

Figure 18-20 Measurement Types in the Components Window



You drag a measurement type on to a BPEL process activity in the designer for initial creation. Measurements are defined as floaters on top of read-only activities in the BPEL process. You can edit the measurement later in the Property Inspector or by double-clicking the measurement. The measurement floaters can be moved around by mouse on top of the BPEL process to achieve the necessary topology.

Each measurement type includes two tabs:

- **General** tab: For defining the impacted activity, the evaluation event that triggers the measurement being taken, the measurement description, and whether the measurement is enabled.
- **Business Indicator** tab: For selecting the business indicators for the measurement.

How to Define a Counter Mark

You can define a counter mark measurement.

To define a counter mark:

1. Access analytics view in a BPEL process as described in How to Access Analytics View.
2. From the Components window, drag a **Counter Mark** icon on to an activity or right-click an activity and select **Counter Mark**.
3. Double-click the icon that is added.

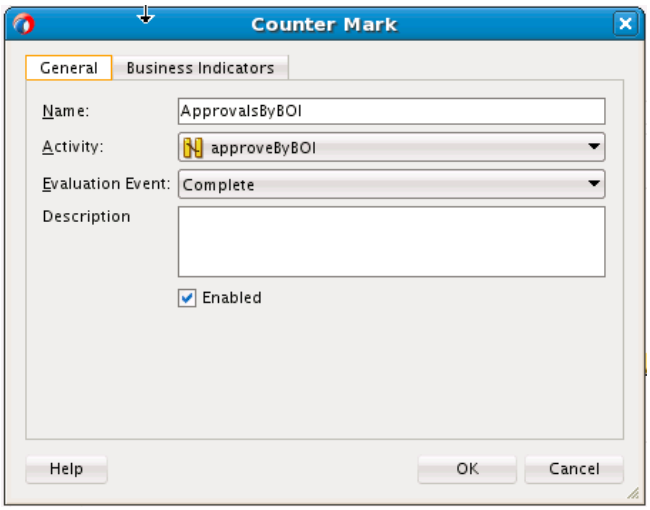
   The Counter Mark dialog is displayed.
4. Enter values appropriate to your environment, and click **OK**. Table 18-4 provides details.

Table 18-4 Counter Mark Dialog - General Tab

Element	Description
Name	Enter the name of the counter mark.
Activity	Displays the BPEL activity on which the counter mark is taken.
Evaluation Event	Select the specific activity event that triggers the counter mark. It can be one of the five activity events: **Activate**, **Compensate**, **Complete**, **Fault**, and **Retry**.
Description	Enter an optional description of the counter mark.
Enabled	Select whether to enable the counter mark. By default, this measurement is enabled.

When complete, the **General** tab of the Counter Mark dialog looks as shown in Figure 18-21.

Figure 18-21 General Tab of Counter Mark Dialog



5. Click the **Business Indicators** tab.

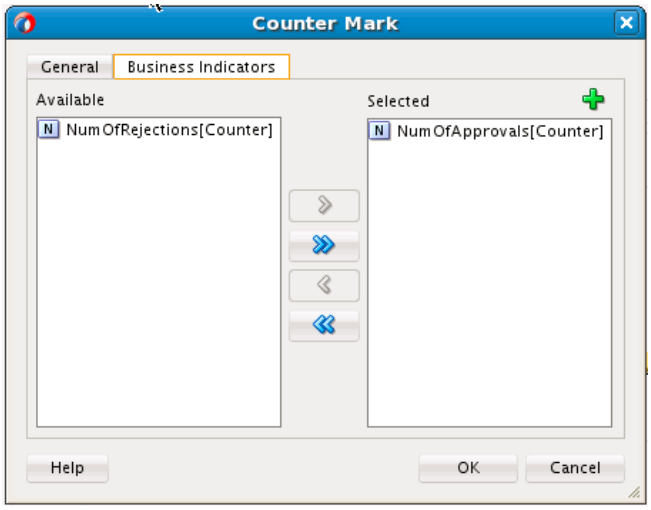
6. Move selected business indicators to the **Selected** section. You can also click the **Add** icon to create new business indicators. Created business indicators are automatically added to the **Selected** section.

Note:

You can only create and select counters for counter marks. Dimensions are implicitly added to counter marks, and you cannot create and select measures for counter marks. Measures can only be created and selected for interval starts, interval stops, and single marks.

When complete, the Business Indicators tab looks as shown in Figure 18-22.

Figure 18-22 Business Indicators Tab



How to Define an Interval Start

You can define an interval start measurement.

To define an interval start:

1. Access analytics view in a BPEL process as described in How to Access Analytics View.
2. From the Components window, drag an **Interval Start** icon on to an activity or right-click an activity and select **Interval Start**.
3. Double-click the icon that is added.

   The Interval Start dialog is displayed.
4. Enter values appropriate to your environment, and click **OK**. Table 18-5 provides details.

Table 18-5 Interval Start Dialog - General Tab

Element	Description
Name	Enter the name of the interval start.

Table 18-5 (Cont.) Interval Start Dialog - General Tab

Element	Description
Activity	Displays the BPEL activity from which the interval starts.
Evaluation Event	Select the specific activity event that triggers the start of the interval. It can be one of the five activity events: **Activate**, **Compensate**, **Complete**, **Fault**, and **Retry**.
Description	Enter an optional description of the interval start.
Enabled	Select whether to enable the interval start. By default, this measurement is enabled.

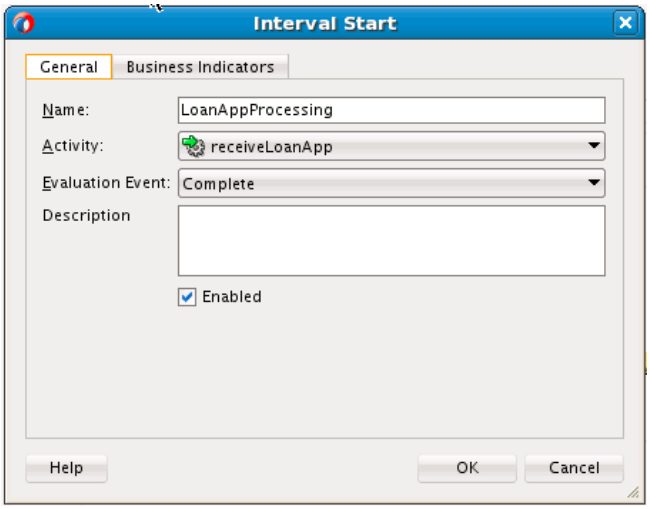
Note:

Any name change is propagated to the corresponding interval stop measurement because the interval name is shared by both the interval start and the interval stop measurements.

In addition, any activity change updates the activity anchor of the interval start floater in the designer.

When complete, the **General** tab of the Interval Start dialog looks as shown in Figure 18-23.

Figure 18-23 General Tab of Interval Start Dialog



5. Click the **Business Indicators** tab.
6. Move selected business indicators to the **Selected** section. You can also click the **Add** icon to create new business indicators. Created business indicators are automatically added to the **Selected** section.

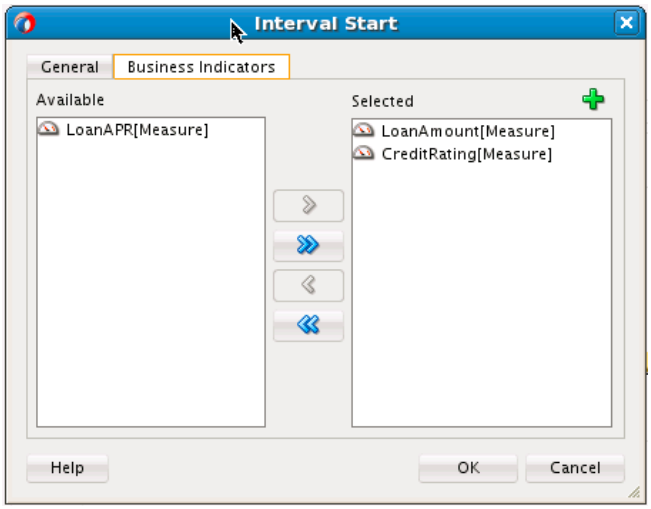
Note:

You can only create and select measures for interval starts. Dimensions are implicitly added to interval starts, and you cannot create and select counters for interval starts. Counters can only be created and selected for counter marks.

Any change on the business indicators for an interval start is propagated to its corresponding interval stop because both the interval start and stop share the same business indicators.

When complete, the **Business Indicators** tab looks as shown in Figure 18-24

Figure 18-24 Business Indicators Tab



How to Define an Interval Stop

You can define an interval stop measurement.

To define an interval stop:

1. Access analytics view in a BPEL process as described in How to Access Analytics View.
2. From the Components window, drag an **Interval Stop** icon on to an activity or right-click an activity and select **Interval Stop**.
3. Double-click the icon that is added.

   The Interval End dialog is displayed.
4. Enter values appropriate to your environment, and click **OK**. Table 18-6 provides details.

Table 18-6 Interval End Dialog - General Tab

Element	Description
Name	Enter the name of the interval stop.

Table 18-6 (Cont.) Interval End Dialog - General Tab

Element	Description
Activity	Displays the BPEL activity on which the interval stops.
Evaluation Event	Select the specific activity event that triggers the stop of the interval. It can be one of the five activity events: **Activate**, **Compensate**, **Complete**, **Fault**, and **Retry**.
Description	Enter an optional description of the interval stop.
Enabled	Select whether to enable the interval stop. By default, this measurement is enabled.

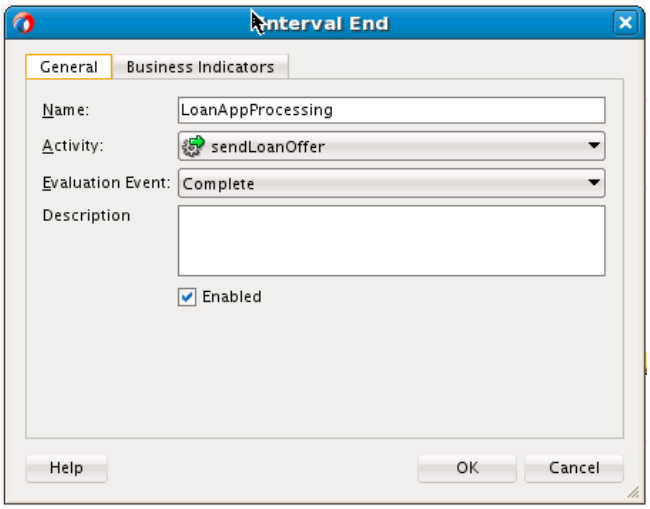
Note:

A name change is propagated to its corresponding interval start, because the interval name is shared by both the interval start and the interval stop measurements.

An activity change updates the activity anchor of the interval stop floater in the designer.

When complete, the **General** tab of the Interval End dialog looks as shown in Figure 18-25.

Figure 18-25 General Tab of Interval End Dialog



5. Click the **Business Indicators** tab.
6. Move selected business indicators to the **Selected** section. You can also click the **Add** icon to create new business indicators. Created business indicators are automatically added to the **Selected** section.

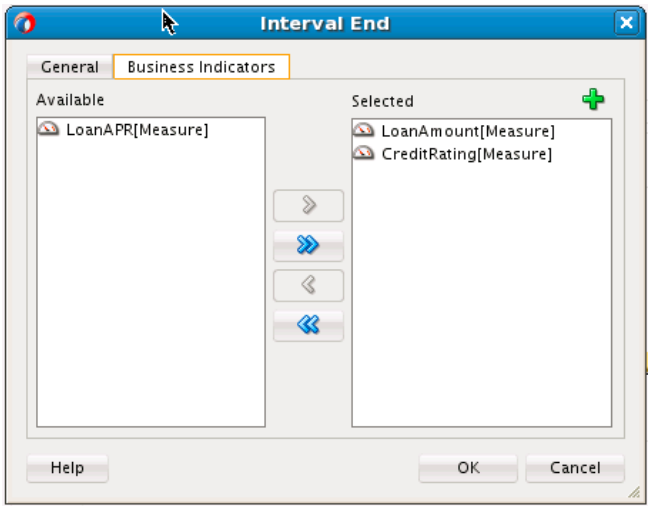
Note:

You can only create and select measures for interval stops. Dimensions are implicitly added to interval stops, and you cannot create and select counters for interval stops. Counters can only be created and selected for counter marks.

Any change on the business indicators for an interval stop is propagated to its corresponding interval start, because both the interval start and stop share the same business indicators.

When complete, the **Business Indicators** tab looks as shown in Figure 18-26.

Figure 18-26 Business Indicators Tab



How to Define a Single Mark

You can define a single mark measurement.

To define a single mark:

1. Access analytics view in a BPEL process as described in How to Access Analytics View.
2. From the Components window, drag a **Single Mark** icon on to an activity or right-click an activity and select **Single Mark**.
3. Double-click the icon that is added.

   The Management Mark dialog is displayed.
4. Enter values appropriate to your environment, and click **OK**. Table 18-7 provides details.

Table 18-7 Management Mark Dialog - General Tab

Element	Description
Name	Enter the name of the single mark.

Table 18-7 (Cont.) Management Mark Dialog - General Tab

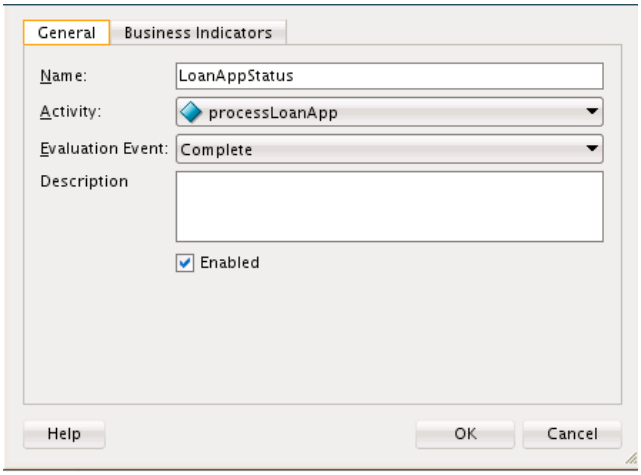
Element	Description
Activity	Displays the BPEL activity on which the single mark is taken.
Evaluation Event	Select the specific activity event that triggers the single mark. It can be one of the five activity events: **Activate**, **Compensate**, **Complete**, **Fault**, and **Retry**.
Description	Enter an optional description of the single mark.
Enabled	Select whether to enable the single mark. By default, this measurement is enabled.

Note:

The activity change updates the activity anchor of the single mark floater in the designer.

When complete, the **General** tab of the Management Mark dialog looks as shown in Figure 18-27.

Figure 18-27 General Tab of Management Mark Dialog



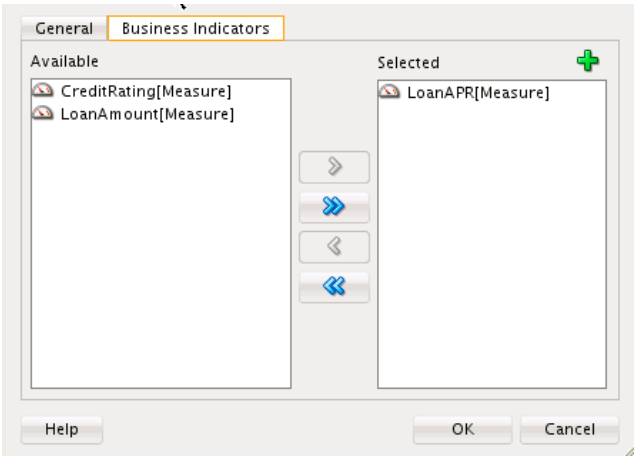
5. Click the **Business Indicators** tab.
6. Move selected business indicators to the **Selected** section. You can also click the **Add** icon to create new business indicators. Created business indicators are automatically added to the **Selected** section.

Note:

The activity change updates the activity anchor of the single mark floater in the designer.

When complete, the **Business Indicators** tab looks as shown in Figure 18-28.

Figure 18-28 Business Indicators Tab



Note:

You can only create and select measures for single marks. Dimensions are implicitly added to single marks, and you cannot create and select counters for single marks. Counters can only be created and selected for counter marks.

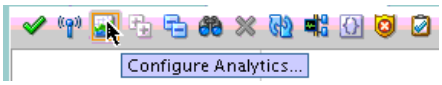
How to Configure Composite-Level Analytic Sampling Points

You can configure analytic sampling points (process start/stop, activity start/stop) at the SOA composite application level. Composite level configuration applies to all BPEL processes in the composite. For information about configuring analytics at the specific BPEL process level, see How to Configure Process-Level Analytic Sampling Points.

To configure composite-level analytic sampling points:

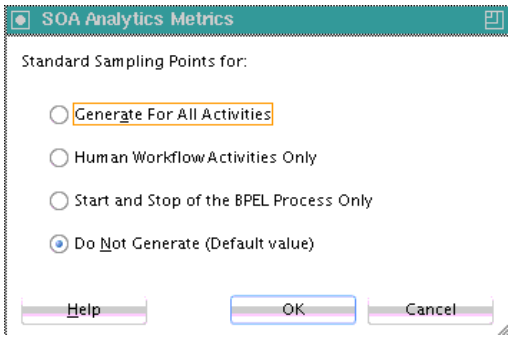
1. Above the SOA Composite Editor, click the **Configure Analytics** icon, as shown in Figure 18-29.

Figure 18-29 Configure Analytics Icon Above SOA Composite Editor



The SOA Analytics Metrics dialog is displayed, as shown in Figure 18-30.

Figure 18-30 SOA Analytics Metrics Dialog



2. Select appropriate options, then click **OK**. Table 18-8 provides details.

Table 18-8 Composite Analytics Setting Dialog

Element	Description
Generate For All Activities	Generates standard analytic events for all process and activity events.
Human Workflow Activities Only	Generates standard analytic events only for human task events.
Start and Stop of the BPEL Process Only	Generates standard analytic events for starting and stopping of the BPEL process.
Do not Generate	Does not generate any standard analytic events.

How to Configure Process-Level Analytic Sampling Points

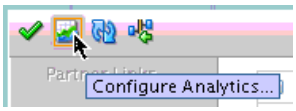
You can configure analytic sampling points (process start/stop, activity start/stop) at the individual BPEL process level. Process level configuration only applies to the generation of standard analytics events for the specific BPEL process. It does not impact the generation of user-defined measurement events for the process. User-defined measurements always generate their measurement events, if enabled.

For information about configuring analytics at the SOA composite application level, see How to Configure Composite-Level Analytic Sampling Points.

To configure process-level analytic sampling points:

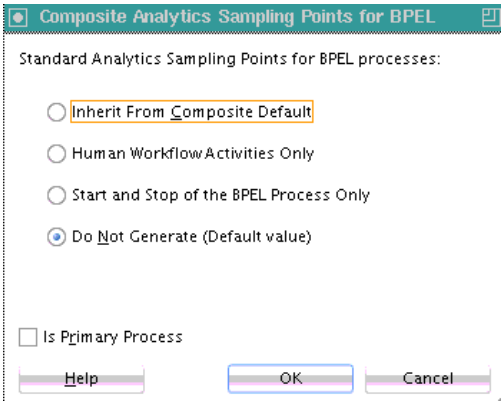
1. Access the BPEL process in analytics view as described in How to Access Analytics View.
2. Above the BPEL process, click the **Configure Analytics** icon, as shown in Figure 18-31.

Figure 18-31 Configure Analytics Icon



The Composite Analytics Sampling Points for BPEL dialog is displayed, as shown in Figure 18-32.

Figure 18-32 Composite Analytics Sampling Points for BPEL Dialog



3. Select appropriate options, then click **OK**. Table 18-9 provides details.

Table 18-9 Process Analytics Setting Dialog

Element	Description
Inherit From Composite Default	Inherits the analytics setting from the composite level analytics configuration described in How to Configure Composite-Level Analytic Sampling Points.
Human Workflow Activities Only	Generates standard analytic events for human task activity events.
Generate For Interactive(s) Only	Generates standard analytic events only for interactive process and activity events such as human task events.
Start and Stop of the BPEL Process Only	Generates standard analytics events for the starting and stopping of the BPEL process.
Do not Generate	Does not generate any standard analytic events.
Is Primary Process	Select to inform analytics runtime if the process is one of the primary processes for the SOA composite application.

How to Edit Business Indicators in the Business Indicator Overview Editor

You can create, edit, and delete business indicators for the SOA composite application in the Business Indicator Overview Editor, regardless of whether or how these business indicators are bound to specific BPEL processes. This editor does not change the bindings for those business indicators as long as they are not deleted. When a business indicator is deleted, all its bindings with the specific BPEL processes are also deleted.

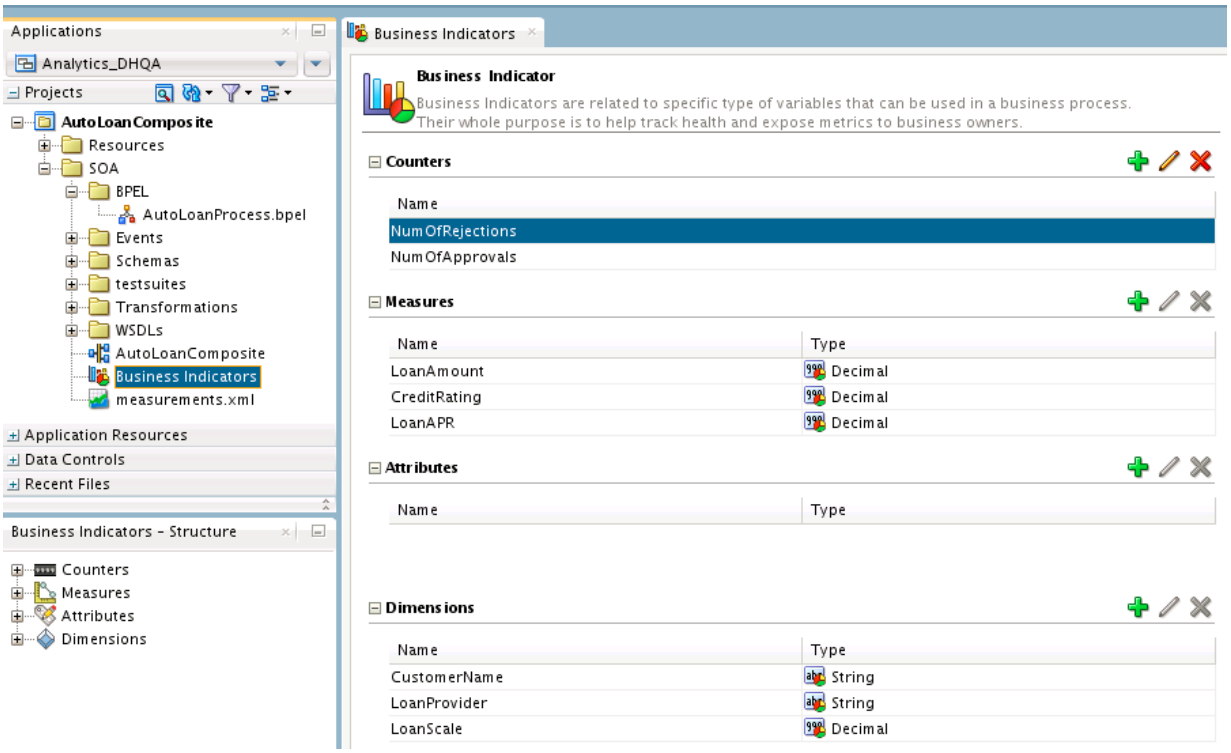
The Business Indicator Overview Editor is the only way to edit and delete business indicators. From the various dialogs for counters, dimensions, and measures that you access from the Structure window or Property Inspector, you cannot edit or delete the business indicators. You can only edit their bindings to the BPEL process. The view of business indicators from the Structure window or Property Inspector is actually a binding view of the business indicators, and not a view of all the business indicators. Any unbound business indicators do not show up from the Structure window or Property Inspector.

Any relevant change in the Business Indicator Overview Editor is reflected in the Structure window or Property Inspector. Any relevant change from the Structure window or Property Inspector is reflected in the Business Indicator Overview Editor.

To edit business indicators in the Business Indicator Overview Editor:

1. In the Applications window, double-click **Business Indicators**. Figure 18-33 provides details.

Figure 18-33 Business Indicators Overview Editor



2. Create, edit, and delete business indicators for counters, measures, and dimensions, as required.

Note:

You can also create a special type of business indicator called an attribute. However, Oracle SOA Suite analytics design time does not currently support attribute binding to BPEL processes. Therefore, you cannot create or bind attributes to the BPEL process in the Structure window or Property Inspector.

Deploying BPEL Analytics

Analytic configurations are included with SOA composite application deployment. If there are no analytics defined in the composite, no deployment of analytics occurs.

The SOA analytics deployment performs the following procedures:

- Populates the analytics definition (composite, process, activity, role) data objects.
- Creates the composite-specific physical and logical data objects (process and activity).

Analytics deployment is divided into two steps based on whether the data population is at the composite level or the component (BPEL process) level:

- Composite-level analytics deployment

  The composite definition data object is populated and the composite-specific physical and logical data objects are created (process and activity). Composite-level

analytics deployment is invoked at composite deployment time. This deployment step is performed once for a composite.

- Component-level analytics deployment

  The process, activity, and role definition data objects are populated. Component-level analytics deployment is invoked at component deployment time. This deployment step is performed for each component of the composite.

Viewing BPEL Analytics at Runtime

The measurement events based on the deployed analytics metadata are triggered. BPEL process and activity events such as start and stop trigger the measurement events. A measurement event captures the values of all business indicators defined for the measurement from the BPEL process service engine, and can be synthesized and published to Oracle BAM.

BPEL process and activity events themselves can also be published to Oracle BAM based on analytics sampling control. BPEL process and activity events also capture applicable business intelligence values.

BPEL measurement events are published to SOA analytics data objects (process and activity) in BAM.

For information about BPEL process analytics integration with Oracle Business Activity Monitoring (BAM), see Chapter "Integrating with Oracle SOA Suite" of *Monitoring Business Activity with Oracle BAM* and Chapter "Gaining Business Insights with Oracle Business Activity Monitoring" of *Understanding Oracle SOA Suite*.

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:

19

Getting Started with Oracle Mediator

This chapter describes Oracle Mediator, which provides transformation, validation, and routing logic to Oracle SOA Suite applications. This chapter also describes how to create a Mediator component and the associated WSDL documents in Oracle JDeveloper.

This chapter includes the following sections:

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.

Mediator provides a lightweight framework to mediate between various components within a composite application, such as business processes, human workflows, and so on, using a Web Services Description Language (WSDL) document as the interface. Mediator converts data to facilitate communication between different interfaces exposed by different components that are wired to build a SOA composite application. For example, 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.

Mediator facilitates integration between events and services, where service invocations and events can be mixed and matched. You can use a Mediator service component to consume a business event or receive a service invocation. A Mediator service component can evaluate routing rules, perform transformations, validate, and either invoke another service or raise another business event. You can use a Mediator service component to handle returned responses, callbacks, faults, and timeouts.

Mediator Functionality

The following sections describe the primary functions that Oracle Mediator supplies to an Oracle SOA Suite application.

Content-Based and Header-Based Routing

Mediator enables you to define rules based on the 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, 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 How to Access Headers for Filters and Assignments.

Synchronous and Asynchronous Interactions

Mediator supports both 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 to raise an event or start a process.

Mediator also supports event-based interactions. Events are one-way (fire-and-forget) asynchronous interactions.

For more information about synchronous and asynchronous interactions, see How to Configure Response Messages and Understanding Message Exchange Patterns of an .

Sequential and Parallel Routing of Messages

Mediator lets you specify that a routing rule be executed either in parallel or in sequence. 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 How to Specify Sequential or Parallel Execution.

Message Resequencing

When you use the Mediator resequencer, it rearranges streams of related but out-of-sequence messages into their sequential order based on the type of resequencer used and the rules you define. When incoming messages arrive in a random order, the resequencer orders the messages based on sequential or chronological information, and then sends the messages to the target services in the correct order based on the resequencing configuration.

For more information about resequencing messages, see Resequencing in .

Data Transformation

Mediator lets you define 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 an XML schema that is compatible with a database.

For more information about transformations, see How to Create XSLT Transformations.

Payload Validation

You can configure Mediators to validate the incoming message payload using a Schematron or an XSD file. You can specify Schematron files for each inbound message part and Mediator executes Schematron file validations for those parts.

For more information about validations, see Specifying Validation and Priority Properties, How to Use Semantic Validation, and `http://www.schematron.com/`.

Java Callouts

Mediator lets you add Java callouts to the routing rules. Java callouts enable you to use external Java classes to manipulate messages flowing through the Mediator.

For more information about Java callouts, see How to Use Java Callouts.

Event Handling

An event is a message sent because an activity occurred in a business environment. Mediator can both subscribe to and raise 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 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 dynamic routing rules using the Mediator Editor.

For more information about dynamic routing, see How to Create Dynamic Routing Rules.

Error Handling

Mediator supports both manual error handling and error handling based on fault policies. 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 Using Error Handling .

Sending Messages Back to the Caller (Echo)

Mediator can echo source messages back to the initial caller without routing the message to another target. Mediator can perform transformations, validations, assignments, or sequencing operations before echoing the message back to the caller.

For more information about Mediator echo support, see "To echo a service:" of How to Specify Mediator Services or Events.

Multiple Part Messages

Mediator can process messages that consist 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 Working with Multiple Part Messages in .

Creating a Mediator

You can create a Mediator in multiple ways, depending on where you are in your application development process. Follow the appropriate instructions in the following sections to create the component.

How to Create a Mediator

You can create a Mediator in a SOA composite application in Oracle JDeveloper at any of the following points in the development cycle:

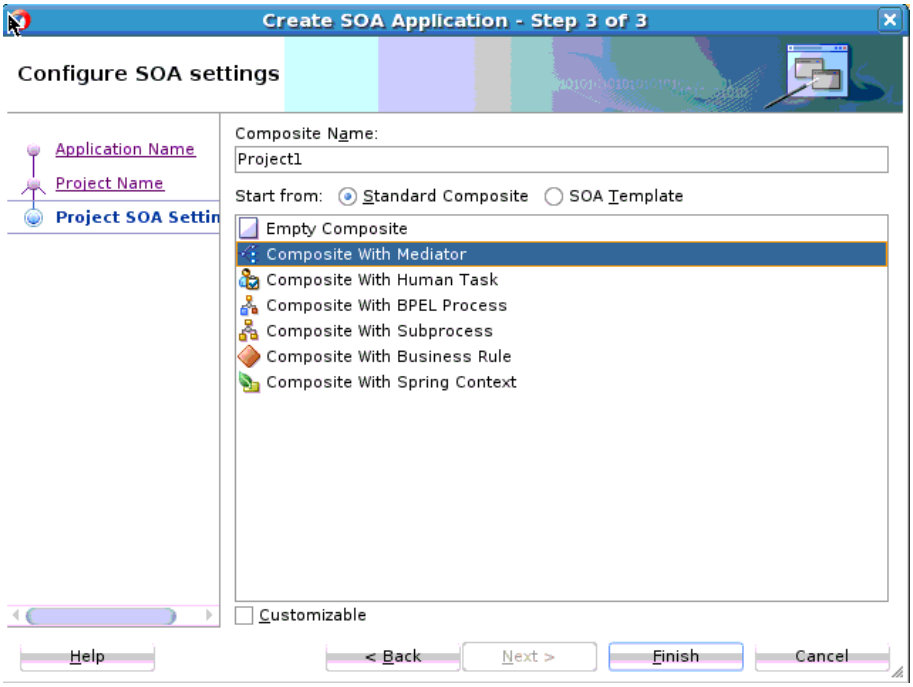
- When you create a composite application
- When you modify an existing composite application
- When you create a project
- When you modify an existing project

When you create a Mediator, the Create Mediator dialog appears so you can name the Mediator and select a template for the interface.

To Create a Composite Application with a Mediator:

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 19-1.

Figure 19-1 Composite with Mediator Selection in Create SOA Project Wizard



3. Click **Finish**.

   The Create Mediator dialog appears.

4. Configure the Mediator interface, as described in Configuring the Mediator Interface Definition.

5. Define routing rules for the Mediator, as described in Creating Routing Rules .

To Create a Mediator in an Existing Composite Application:

1. Open the composite application to which you are adding a Mediator in the SOA Composite Editor.

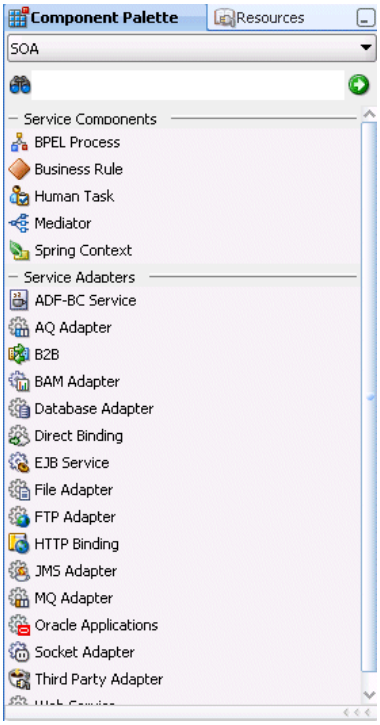
2. Drag and drop a Mediator from the Components window (shown in Figure 19-2) to the **Components** section of the editor.

   Alternatively, right-click a blank area in the Components section of the editor. Select **Insert > Mediator** from the context menu that appears.

   Tip:

   The Components window is to the right of the SOA Composite Editor.

Figure 19-2 Components Window with a Mediator Service Component



The Create Mediator dialog appears.

3. Configure the Mediator interface, as described in Configuring the Mediator Interface Definition.

4. Define routing rules for the Mediator, as described in Creating Routing Rules .

To create a new project with a Mediator:

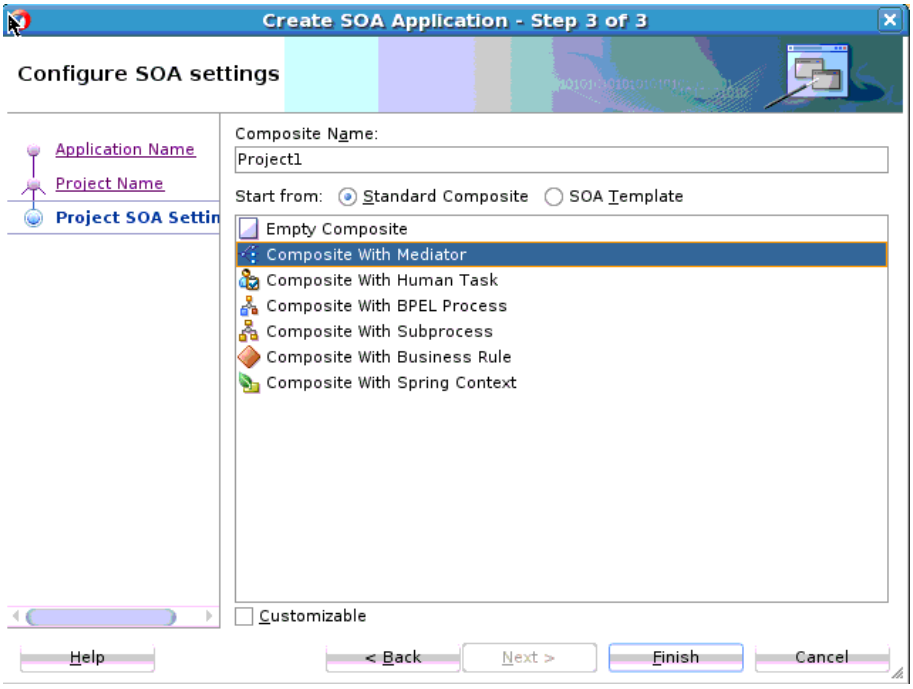
1. Right-click in the Applications window, and then select **New**.

   The New Gallery wizard appears.

2. Create and name a new SOA project in the SOA Tier category.

3. On the Configure SOA Settings page of the New Gallery dialog, select **Composite With Mediator** from the **Composite Template** list, shown in Figure 19-3.

Figure 19-3 Create SOA Project Wizard with Composite With Mediator Template Shown



4. Click **Finish**.

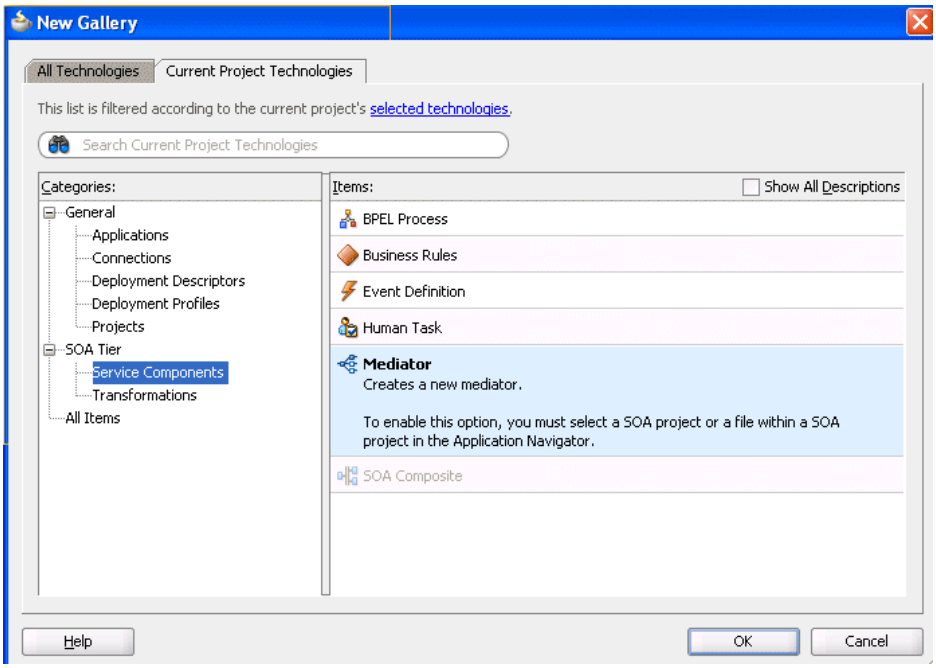
   The Create Mediator dialog appears.

5. Configure the Mediator interface, as described in Configuring the Mediator Interface Definition.

To create a Mediator in an existing project:

1. In the Applications window, select the project to which you want to add a 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 19-4.

Figure 19-4 New Gallery Dialog with Mediator Service Component



4. Click **OK**.

   The Create Mediator dialog appears.

5. Configure the Mediator interface, as described in Configuring the Mediator Interface Definition.

6. Define routing rules for the Mediator, as described in Creating Routing Rules .

Introduction to the Mediator Editor Environment

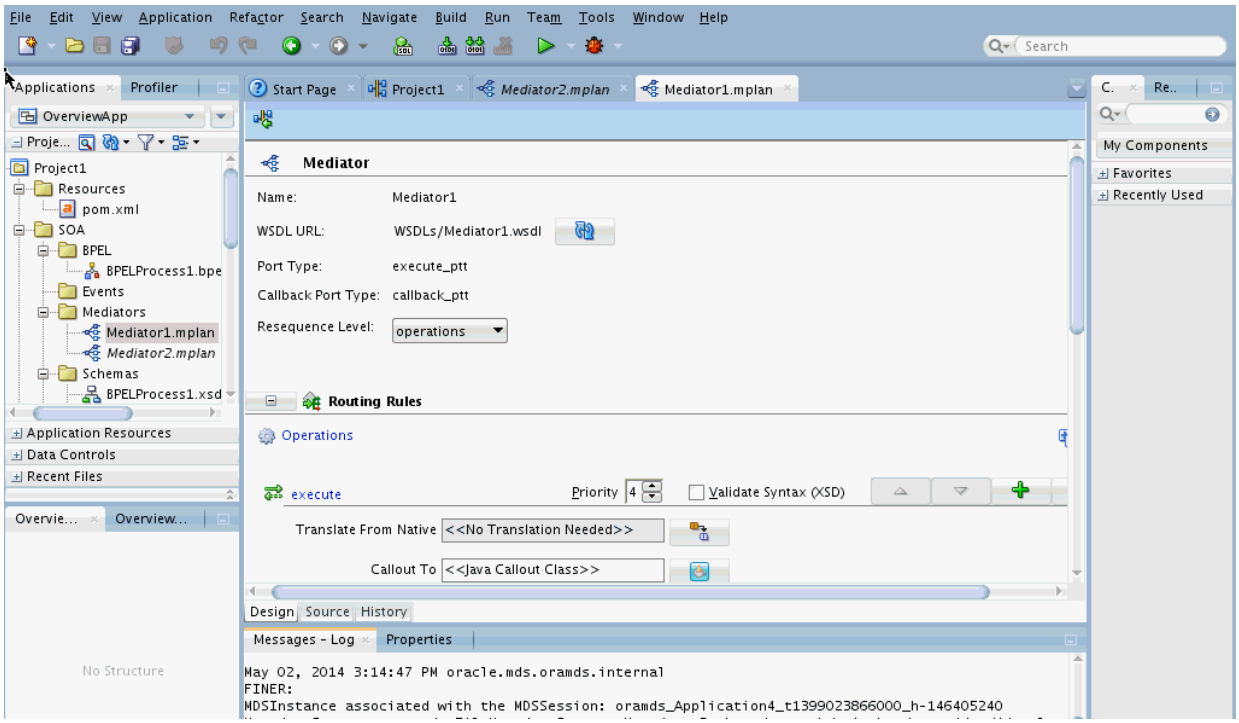
You can create a Mediator service component in a SOA composite application of Oracle JDeveloper and then configure it using the Mediator Editor. To display the Mediator Editor, double-click the Mediator service component in the . For information about the , see Getting Started with Developing SOA Composite Applications.

Figure 19-5 shows the Mediator Editor along with the Applications window, Structure, and Messages windows.

Note:

Oracle recommends using a Unicode database with `AL32UTF8` as the database character set for full globalization support in Mediator.

Figure 19-5 Mediator Editor Window



Each section of the view shown in Figure 19-5 lets you perform specific design and deployment tasks. The sections in this view include the following:

- Applications window

  The Applications window, shown in the upper left section of Figure 19-5, displays the Mediator mplan file. This file appears under the SOA Content folder of the project where you created a Mediator. For more information about the Applications window and the composite files, see Table 2-4.

- Mediator Editor

  The Mediator Editor, shown in the middle of Figure 19-5, provides a visual view of the Mediator. This view appears when you perform one of the following actions:

  – Double-click an icon in the .

  – Double-click the `.mplan` file for the Mediator in the Applications window.

- Source View

  The Source view displays the source code of a Mediator. Click **Source** at the bottom of the Mediator Editor to view the source code. The code in Source view is immediately updated to reflect any changes to an a Mediator.

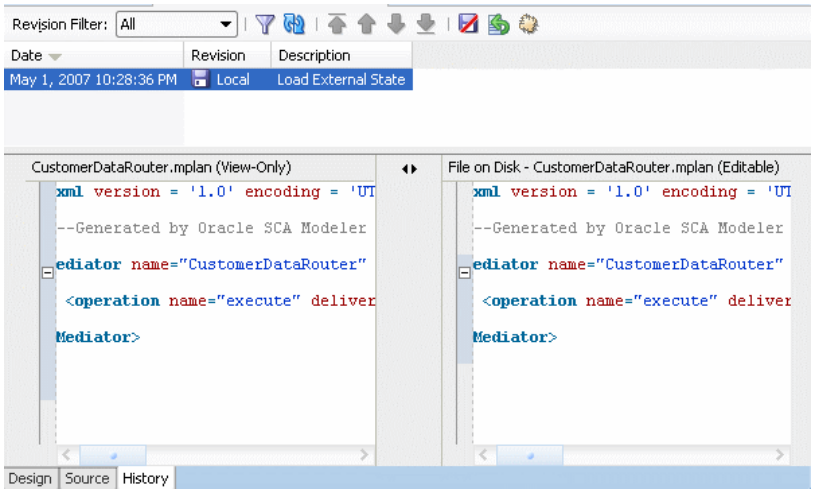
  The following example shows sample Mediator source code:

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

- History Window

The History window displays history information about the 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 19-5 to open the History window. Figure 19-6 shows the History view for a Mediator file.

Figure 19-6 History Window



Configuring the Mediator Interface Definition

When you create a new Mediator, you can specify an interface template that generates a basic set of default files in the Mediator project. These files provide a framework from which you can design and configure the Mediator. You can create a Mediator with the following interface options:

- **Mediator with no interface definition**

  This creates an empty Mediator and does not create a WSDL file. This method provides you with the flexibility to create the SOA components in the order you want.

  After you create a Mediator without an interface definition, you must create a service or an event that starts the component. You can also define the interface implicitly by dragging and dropping a service, or the output interface from another component, to the Mediator input.

- **Mediator with the interface defined by a WSDL file**

  This bases the interface definition on a WSDL file, which describes the interfaces of a Mediator, such as port type, operations, services, and schemas. The WSDL file can already exist or you can generate one from a schema file.

- **Mediator with a one-way interface**

  This defines an interface with a one-way interaction, where the client sends a message to a service and the service does not need to reply.

- **Mediator with a synchronous interface**

  This creates an interface with 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.

- **Mediator with an asynchronous interface**

  This creates an interface with 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.

- **Mediator that subscribes to events**

  This creates a Mediator that subscribes to a business event 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 Using Business Events and the Event Delivery Network.

  To subscribe to events, the events must be defined in an Event Definition (EDL) file.

How to Configure the Mediator Interface Definition

You configure the interface definition for a Mediator on the Create Mediator dialog.

To configure the Mediator interface definition:

1. Create a Mediator using one of the methods described in Creating a Mediator.

   The Create Mediator dialog appears.

2. In the **Name** field, enter a name for the Mediator service component.

3. Select one of the following from the **Template** list. Refer to the descriptions at the beginning of this section for more information on each.

   - Define Interface Later
   - Interface Definition from WSDL
   - One-Way Interface
   - Synchronous Interface
   - Asynchronous Interface
   - Subscribe to Events

   Figure 19-7 and Figure 19-8 illustrate how the properties change on the Create Mediator dialog for different interface types.

Figure 19-7 Synchronous Interface Template Selection on the Create Mediator Dialog

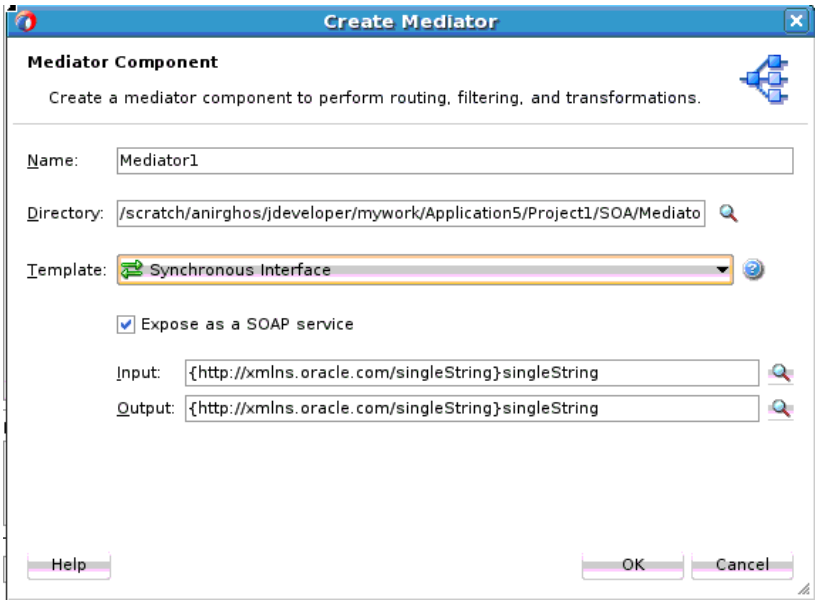
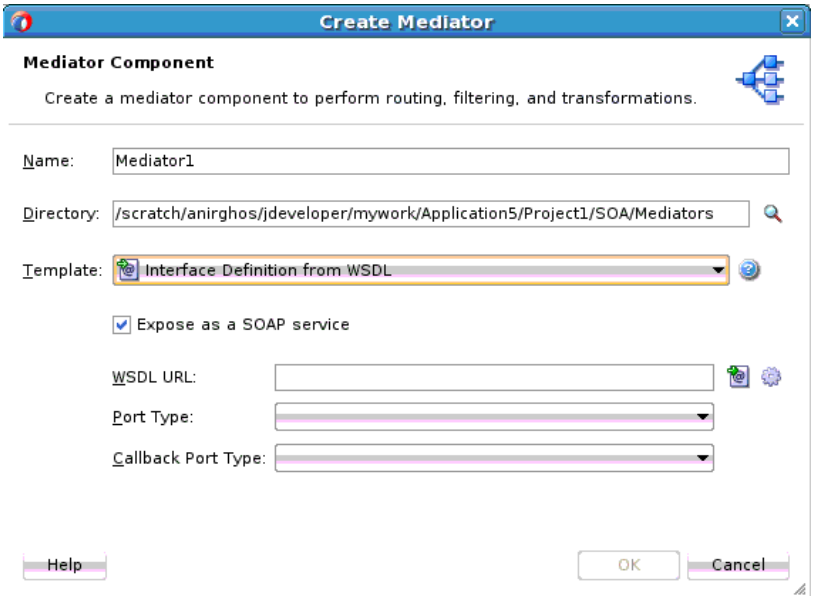


Figure 19-8 Interface Definition from WSDL Template Selection on the Create Mediator Dialog

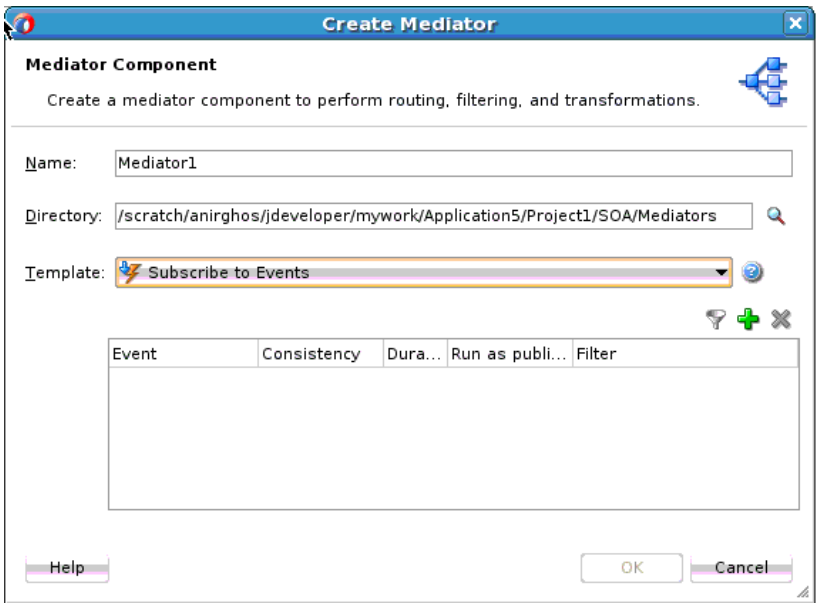


4. For any interface type except **Subscribe to Events**, configure the appropriate properties. For information about the displayed properties for each type, see Table 19-1 following these instructions.

5. If you selected Subscribe to Events, do the following:

   a. Click **Add** on the Create Mediator dialog.

Figure 19-9 Subscribe to Events Template Selection in Create Mediator Dialog



The Event Chooser dialog appears.

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

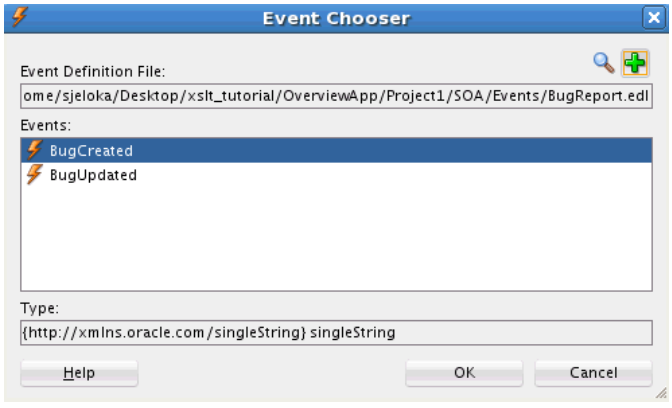
The SOA Resource Browser dialog appears.

c. 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 Using Business Events and the Event Delivery Network.

d. Select one or more events in the **Event** field, as shown in Figure 19-10, and click **OK**.

Figure 19-10 Event Chooser Dialog



e. 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.

f. In the **Run as publisher** field, select whether to run the event subscription under the security of the event publisher.

By default, event subscriptions run under the security of the event publisher.

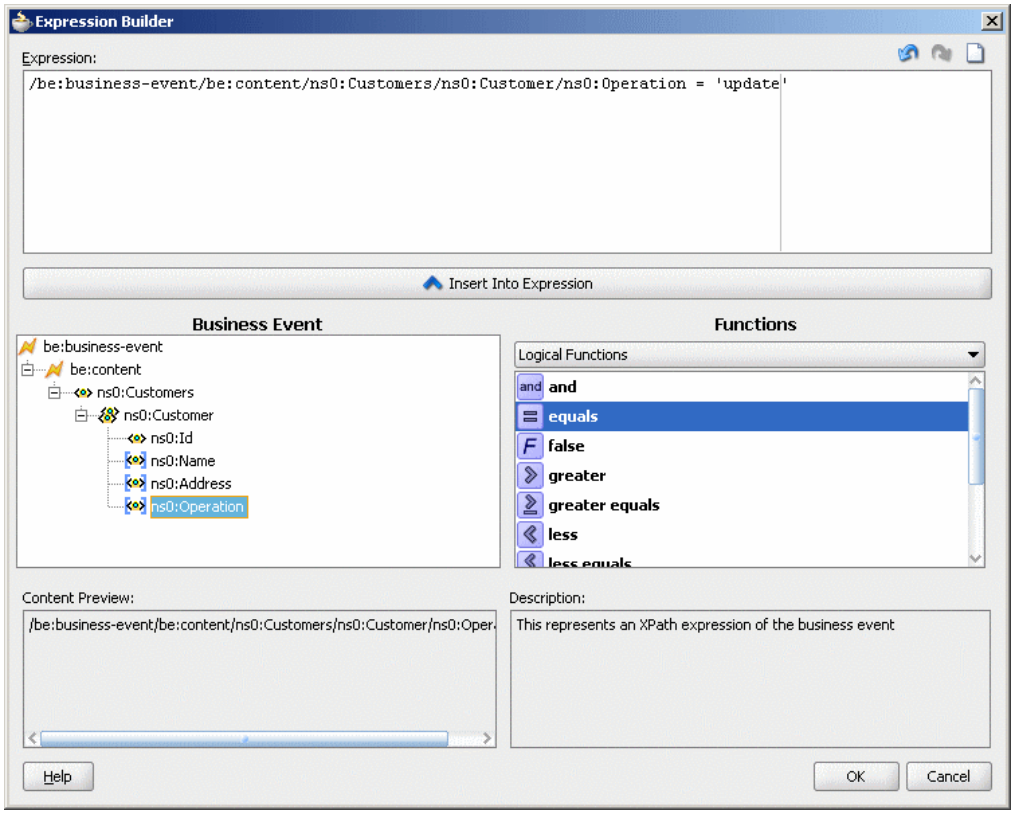
g. To filter the event, double-click the **Filter** column of the selected event, or select the event and then click the **filter** icon (first icon).

The Expression Builder dialog appears.

h. In the **Expression** field, enter an XPath expression and click **OK**.

Figure 19-11 shows a sample Expression Builder dialog.

Figure 19-11 Business Event Filter



The expression you created appears in the **Filter** column of the Create Mediator dialog.

i. Click **OK**.

6. Click **OK** on the Create Mediator dialog.

7. If you chose to create a Mediator without an interface, you must create the interface at a later time as described in How to Define an Interface for a Mediator .

The following table lists and describes the properties you can configure to define an interface. The available properties change depending on the interface type you select, so not all of the listed properties apply to all interface types.

Table 19-1 Mediator Interface Properties

Property	Description
Create Composite Service with SOAP Bindings	Select this option to create an exposed service with SOAP bindings that is automatically connected to your Mediator when the interface is generated.
WSDL URL	Enter the location of the WSDL file to use when creating the interface from a WSDL file. 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 Generating a WSDL File.
Port Type	Select the port type name from the list. The available port types are parsed from the WSDL file that you specify in the **WSDL URL** field.
Callback Port Type	Select the port type name to which the response message is sent in an asynchronous communication. The available port types are parsed from the WSDL file that you specify in the **WSDL URL** field.
Input	Enter the schema element for the input message. Click **Search** to the right of the field to select the element. By default, the **singleString** schema element is selected for the input message. For a sample schema, see the schema that follows after this table.
Output	Enter the schema element for the output message. Click **Search** to the right of the field to select the element. By default, the **singleString** schema element is selected for the input message.

You can use any XSD schema to specify the format of the input document that 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>
```

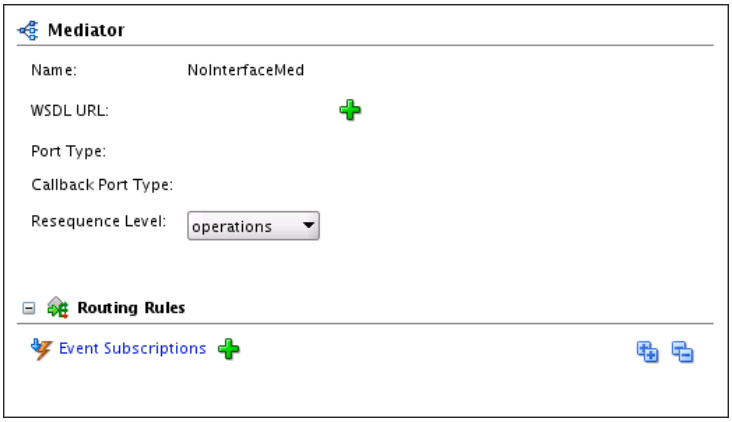
What Happens When You Create a Mediator

The Mediator files are generated under the specified application and project in the Applications window, and the new Mediator appears in the Mediator Editor in Design view. If you created the Mediator with an interface definition and the WSDL file did not already exist, the new WSDL file is also generated with the same name as the Mediator. If the WSDL file you specified is located in a different directory than the project files, the file and its associated schema files are copied to the Mediator project.

Without an Interface Definition

This Mediator has no associated WSDL file, port types, or operations. You must define these separately as described in Defining an Interface for a Mediator. Figure 19-12 shows how a Mediator created with no interface definition appears in the Mediator Editor.

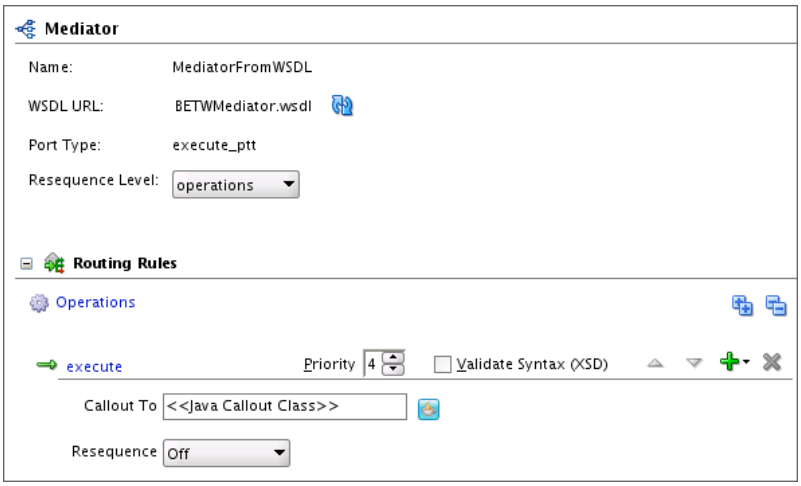
Figure 19-12 Mediator with no Interface Definition in the Mediator Editor



With a WSDL-Based Interface

The appearance and source code of this Mediator varies depending on the name of the WSDL file and the port types and operations defined by the WSDL file. Figure 19-13 shows a sample Mediator created from a WSDL file.

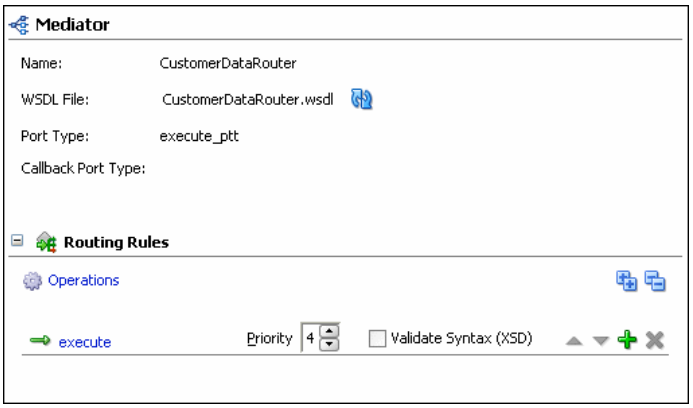
Figure 19-13 Mediator from WSDL in the Mediator Editor



With a One-Way Interface Definition

Figure 19-14 shows how a Mediator created with a one-way interface appears in the Mediator Editor. The arrow to the left of the **execute** operation represents a one-way operation.

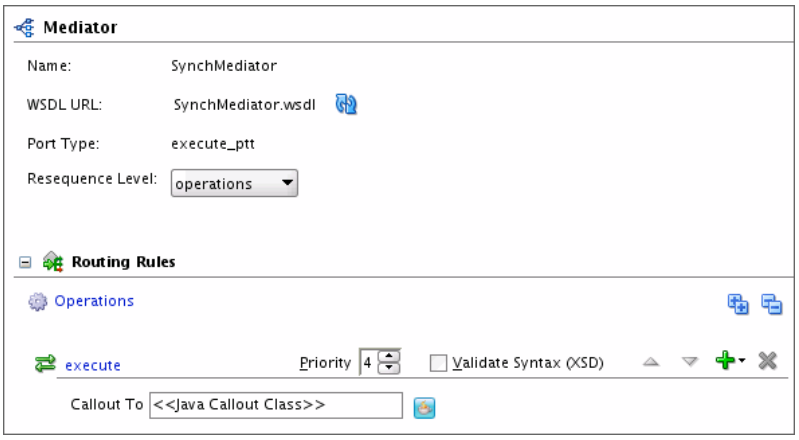
Figure 19-14 One-Way Interface in the Mediator Editor



With a Synchronous Interface Definition

In a synchronous interaction, only one port is defined because the response is sent to the same port as the request. Figure 19-15 shows how a Mediator created with a synchronous interface appears in the Mediator Editor. The arrows to the left of the **execute** operation in Figure 19-15 represent a synchronous operation.

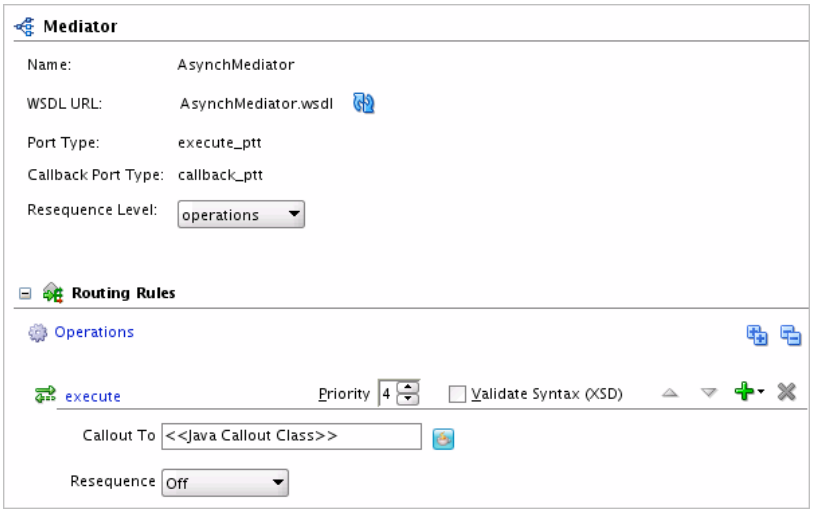
Figure 19-15 Synchronous Mediator in the Mediator Editor



With an Asynchronous Interface Definition

Figure 19-16 shows how a 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 19-16 represent an asynchronous operation.

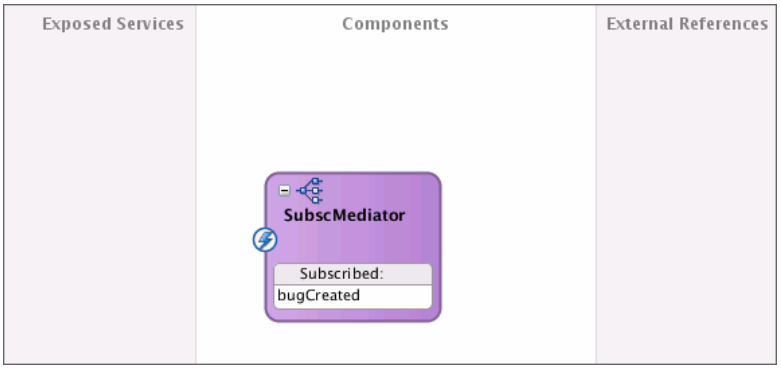
Figure 19-16 Asynchronous Mediator in the Mediator Editor



With an Event Subscription

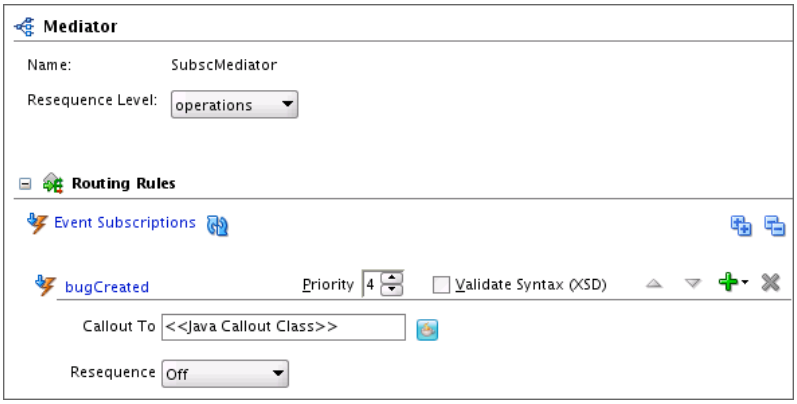
When you view the Mediator in the SOA Composite Editor, the icon on the left side of the Mediator indicates that this Mediator is configured for an event subscription, as shown in Figure 19-17.

Figure 19-17 Mediator Created with the Subscribe to Events Template



When you double-click the Mediator, the Mediator Editor appears, as shown in Figure 19-18.

Figure 19-18 Event Subscription Mediator in the Mediator Editor



Defining an Interface for a Mediator

After you create a Mediator without an interface definition, you must define the interface by subscribing to events or by defining services. You can define services in the following two ways:

- Connect the Mediator to a service through a wire in the SOA Composite Editor.
- Use the **Define Service** or **Add Event Subscription** option in the Mediator Editor.

How to Define an Interface for a Mediator

The following procedures describe how to define an interface for an existing Mediator by subscribing to events, by defining services creating a wire in the composite, and by defining services using the Mediator Editor.

To Subscribe to Events:

To subscribe to events, the events must be defined in an Event Definition (EDL) file.

1. Open the 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 use an existing EDL file, follow the instructions under Configuring the Mediator Interface Definition beginning with Step 55.b.

   Note:

   You can alternatively create a new EDL file. Click **Create EDL file** to create a new EDL file. Enter the event details in the Create Event Definition dialog that appears.
5. Click **OK**.

To Define Services for a Mediator Using a Wire:

- In the , drag a wire from a Mediator to a service.

  For more information about wires and how to wire a service component to a service, see How to Wire a Service and a Service Component.

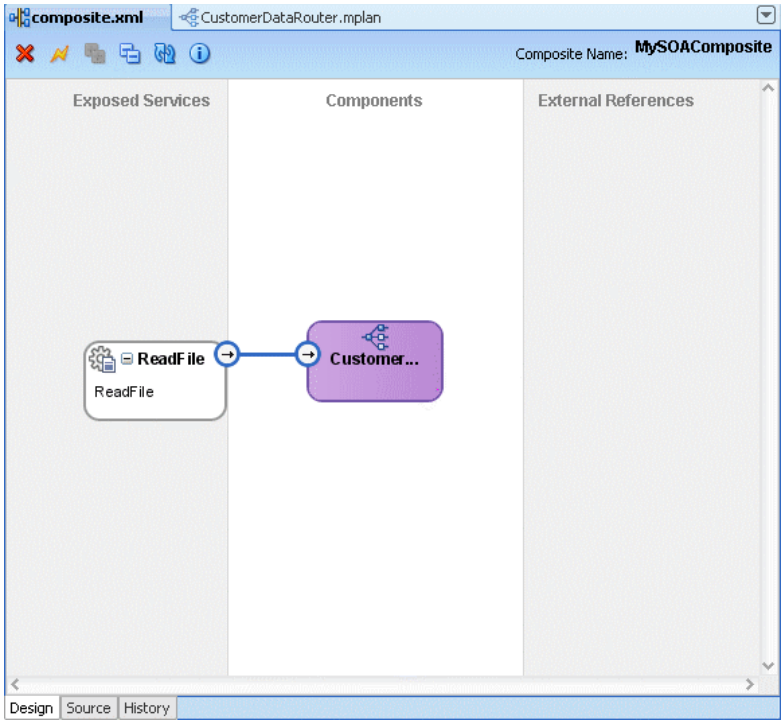
  Note:

  You can also wire a Mediator with a defined service interface to another interface. However, to connect a Mediator to a service, the interface of the Mediator and of the service must match.

When you define a service using a wire, the service for the Mediator is automatically defined using the WSDL file from the wire source. For example, if

you connect the ReadFile service shown in Figure 19-19 to the CustomerDataRouter Mediator, the CustomerDataRouter Mediator automatically inherits the service definition of the ReadFile service.

Figure 19-19 Connecting Mediator to a Service



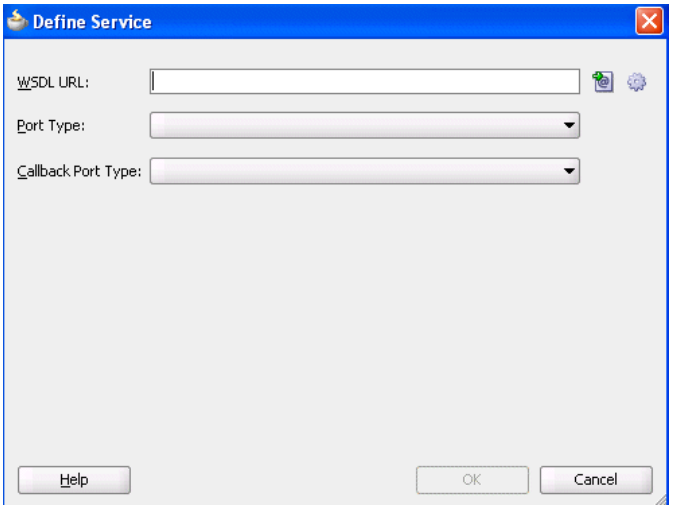
For information about how wiring two Mediator service components can cause an infinite loop, see What You May Need to Know About Adding and Deleting Wires.

To Define Services for a Mediator in the Mediator Editor:

1. Display the 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 19-20.

Figure 19-20 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 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**.

Generating a WSDL File

You can generate the WSDL file for a message using an XML schema definition (XSD) file. When working with Mediator, you can generate a WSDL file at either of the following times:

- When you are creating a 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 a 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.

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 a Mediator or when you are defining a service for a Mediator.

1. On the Create WSDL dialog, accept the default values or enter the following information for the WSDL file:

Table 19-2 WSDL Properties

Property	Description
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 SOA/WSDLs folder under the project folder.

Table 19-2 (Cont.) WSDL Properties

Property	Description
Namespace	A namespace address for the WSDL file; for example, `http://oracle.com/esb/namespaces/Mediator`. The default namespace is based on the JDeveloper application name, project name, and the mediator name. 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`.

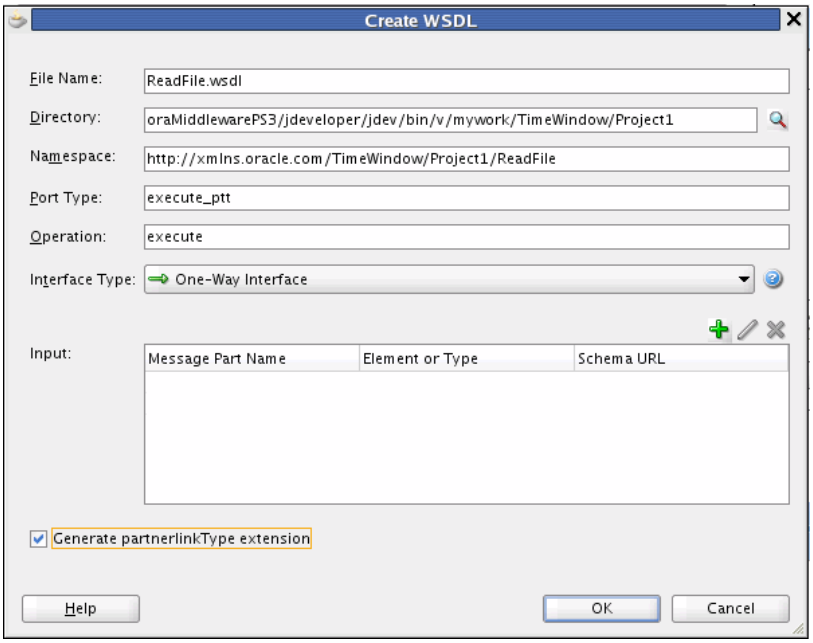
Note:

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 19-21.

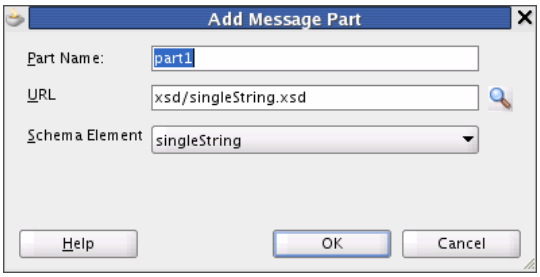
Figure 19-21 Create WSDL Dialog for a One-Way Interface



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 19-22.

Figure 19-22 Add Message Part Dialog

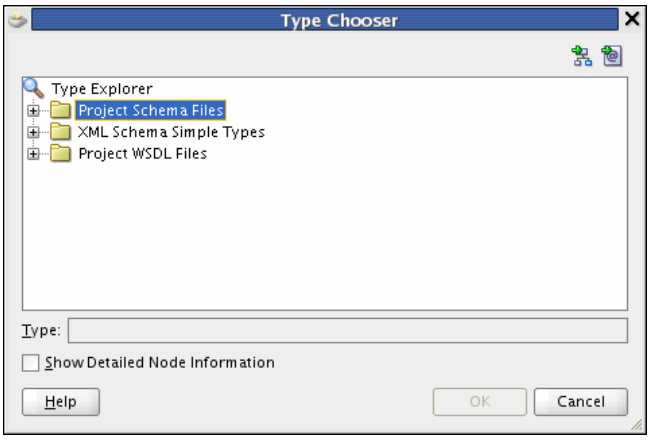


4. In the **Part Name** field, enter a name for the message part.

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 19-23.

Figure 19-23 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.

   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 Mediator.

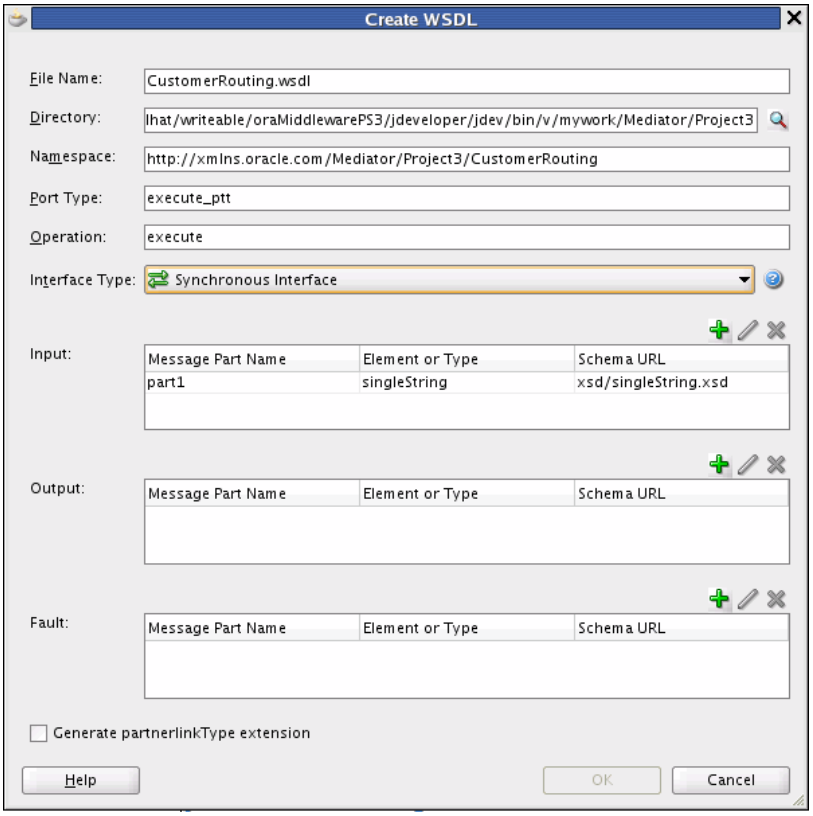
To generate a WSDL file for a synchronous interface from an XSD file:

Perform these steps after the Create WSDL dialog appears when you are creating a Mediator or when you are defining a service for a Mediator.

1. On the Create WSDL dialog, enter the information for the properties listed in Table 19-2.
2. In the **Interface Type** field, select **Synchronous Interface**.

   The **Input**, **Output**, and **Fault** fields appear, as shown in Figure 19-24.

Figure 19-24 Create WSDL Dialog for a Synchronous Interface



3. Repeat steps 3 to 8, as in the previous procedure.
4. Repeat the same 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.
5. Click **OK**.

Note:

Partner link types are generally used in BPEL, so you do not need to select **Generate partnerlinkType extension** for Mediator.

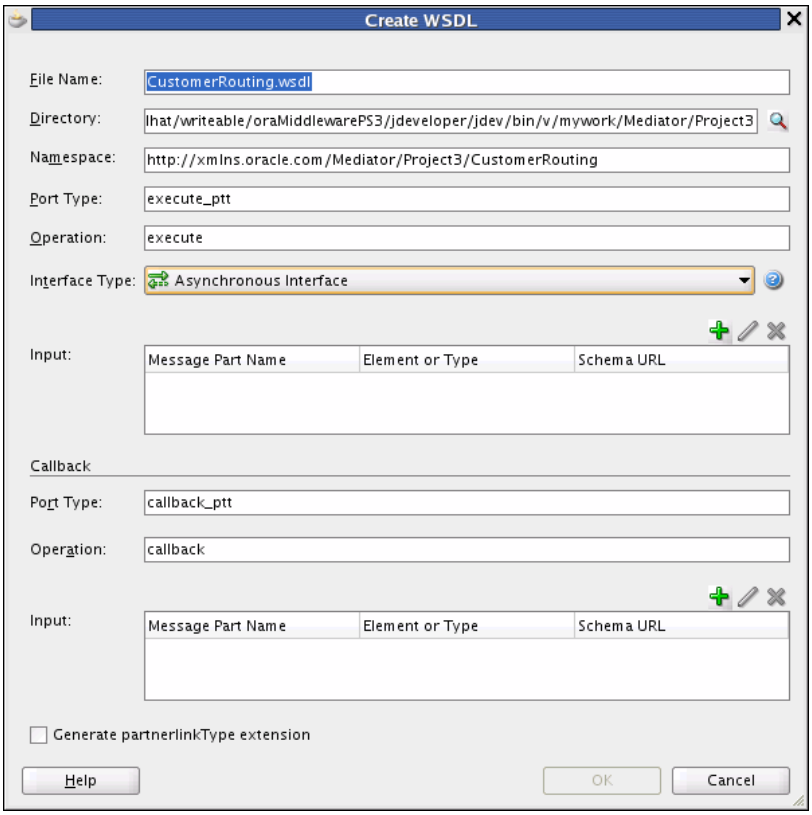
To generate a WSDL file for an asynchronous interface from an XSD file:

Perform these steps after the Create WSDL dialog appears when you are creating a Mediator or when you are defining a service for a Mediator.

1. On the Create WSDL dialog, enter the information for the properties listed in Table 19-2.
2. In the **Interface Type** field, select **Asynchronous Interface**.

   The **Input** field and **Callback** section appear, as shown in Figure 19-25.

Figure 19-25 Create WSDL Dialog for an Asynchronous Interface



3. Repeat steps 3 to 8, as in the earlier procedure.
4. Repeat the same 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.

5. 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.

---

6. Click **OK**.

Note:

Partner link types are generally used in BPEL, so you do not need to select **Generate partnerlinkType extension** for Mediator.

Specifying Validation and Priority Properties

After creating a Mediator, you can configure properties for the operation or event subscription specified for the component. On the Mediator Editor, you can specify whether to validate the schemas of inbound messages and you can specify a priority for the operation or event subscription.

To validate inbound message schemas, select the **Validate Syntax (XSD)** check box for an operation or event subscription in the Routing Rules section of the Mediator Editor. The Mediator Engine validates the XML inbound payload syntactic structure against the associated XML schema. Any syntax error, such as an incorrect element name or location, causes a fault and the routing rule is not processed.

To specify a priority for an component, select a value from zero to nine in the **Priority** field in the Mediator Editor's Routing Rules section. This determines the order in which messages are retrieved for all service components. This property is only valid for parallel routing rules and not sequential. For more information about priorities, see "Basic Principles of Parallel Routing Rules".

Modifying a Mediator Service Component

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

How To Modify Mediator Operations

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

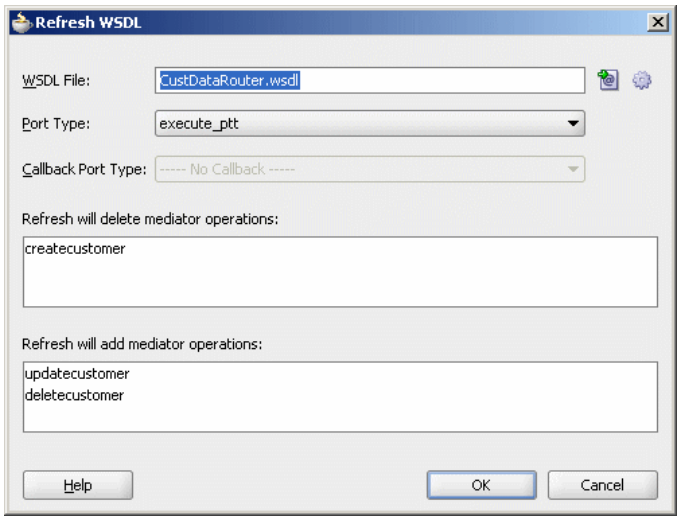
To modify operations:

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 19-26 shows the Refresh WSDL dialog.

Figure 19-26 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 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**.

How To Modify Mediator Event Subscriptions

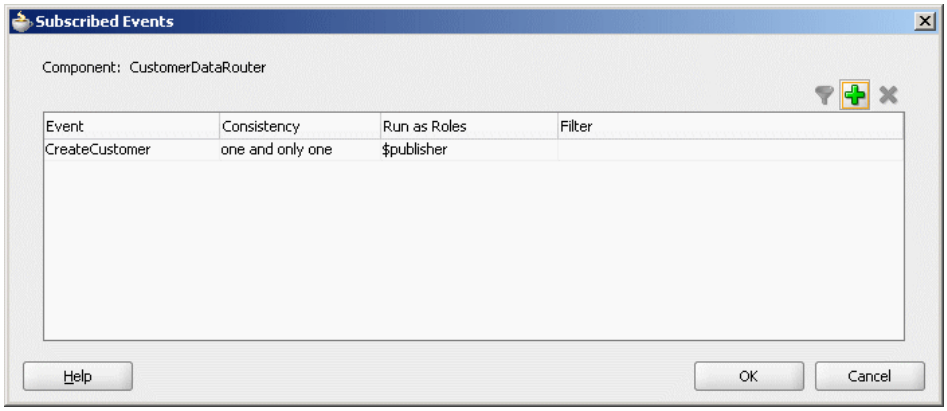
You can subscribe to new events, modify existing event subscriptions, and unsubscribe from subscribed events using the **Manage Event Subscriptions** option in the Mediator Editor.

To modify event subscriptions:

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 19-27.

Figure 19-27 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 How to Configure the Mediator Interface Definition.

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

20

Creating Oracle Mediator Routing Rules

This chapter describes Oracle Mediator routing rules and how to specify routing rules for a Mediator service component. Routing rules include transformation, filtering, validation, mapping, and routing logic.

This chapter includes the following sections:

The following chapter provide additional information about defining routing rules for specific scenarios:

Introduction to Routing Rules

Routing rules are mediation logic or execution logic that you define to achieve the requisite mediation. 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 Mediator. You can use routing rules to specify how a message processed by a Mediator reaches its next destination. Routing rules specify where a Mediator sends the message, how it sends the message, and what changes should be made to the message structure before sending it to the target service.

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. 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 How to Create Static Routing Rules and How to Create Dynamic Routing Rules. For information about standard message exchange patterns and how they are handled by Mediator, see Understanding Message Exchange Patterns of an .

Static Routing Rules

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.

When you define static rules, you can specify the following types of information:

- **Target Service**

  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 How to Specify Mediator Services or Events.

- **Execution Type**

  Mediator executes routing rules either sequentially (that is, running in the same thread) or in parallel (running on different threads). For information about specifying an execution type, see How to Specify Sequential or Parallel Execution.

  Note:

  For synchronous service invocations, the routing rule should always be sequential.

- **Reply, Callback, and Fault Handlers**

  You can define how Mediator handles synchronous reply, callback, and fault messages. For information about handlers, see How to Configure Response Messages, How to Handle Faults, and Static Routing Rule Components.

Types of Static Rules

You can define the following types of static rules for a Mediator:

- **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 How to Specify an Expression for Filtering Messages.

- **Transformations**

  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 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 How to Create XSLT Transformations and How to Assign Values.

- **Accessing Header Variables from Expressions**

  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 How to Access Headers for Filters and Assignments and Manual Expression Building for Accessing Properties for Filters and Assignments.

- **Schematron-Based Validations**

  You can specify the Schematron files that Mediator should use to validate different parts of an inbound message. For information about performing Schematron-based validations, see How to Use Semantic Validation.

- **Java Callouts**

  Mediator lets you add Java callouts to the routing rules. Java callouts enable you to use external Java classes to manipulate messages flowing through the Mediator. For information about using Java callouts, see 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:".

Static Routing Rule Components

Static routing rules define the following components:

- **Request Handler**: Defines how Mediator handles incoming requests.
- **Reply Handler**: Defines how the synchronous response from the called service is handled by Mediator.
- **Fault Handler**: Defines how the named or declared faults from the called service are handled by Mediator.
- **Callback Handler**: Defines how the asynchronous response and callback from the called service are handled by Mediator.
- **Timeout Handler in Callback**: Defines how long Mediator waits for the asynchronous response and callback before performing timeout handling for the particular asynchronous request.
- **Event Publishing and Service Invocation**: Calls other services or publishes an event depending on the configuration of the handlers.

Dynamic Routing Rules

A dynamic routing rule lets you externalize the routing logic to an Oracle Rules Dictionary or Domain Value Map (DVM), which in turn enables dynamic modification of the routing logic in a routing rule. Dynamic routing enables you to dynamically route messages at runtime from a mediator to multiple target services, based on the message content.

Dynamic routing rules are described in more detail in How to Create Dynamic Routing Rules.

Sequential and Parallel Execution

Routing rules can be executed sequentially or in parallel. This section describes the basic principles of both types of execution. If an operation or event has both sequential and parallel routing rules, first sequential routing rules are evaluated and actions are performed, and then parallel routings are queued for parallel execution.

Note:

If a Mediator service component with a request-response interface has only parallel routing rules, the Mediator service component does not send a response back to the caller. Though you can create this type of Mediator service component, the caller of the Mediator service component does not receive a response at runtime.

Basic Principles of Sequential Routing Rules

Mediator processes sequential routing rules based on the following principles:

- Mediator evaluates routings and performs the resulting actions sequentially. Sequential routings are evaluated in the same thread and transaction as the caller.
- Mediator always enlists itself into the global transaction propagated through the thread that is processing the incoming message. For example, if an inbound JCA adapter invokes a Mediator, the Mediator enlists itself with the transaction that the JCA adapter has initiated.
- Mediator propagates the transaction through the same thread as the target components while executing the sequential routing rules.
- Mediator never commits or rolls back transactions propagated by external entities.
- Mediator manages the transaction only if the thread-invoking Mediator does not already have an active transaction. For example, if Mediator is invoked from inbound SOAP services, Mediator starts a transaction and commits or rolls back the transaction depending on success and failure.

Basic Principles of Parallel Routing Rules

Mediator processes routing rules in parallel based on the following principles:

- Mediator queues and evaluates routings in parallel in different threads.

  The messages of each Mediator service component are retrieved in a weighted, round-robin fashion to ensure that all Mediator service components receive parallel processing cycles. This is true even if one or more Mediator service components produce a higher number of messages compared to other components. The weight used is the message priority set when designing a 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 a 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.

- Mediator initiates a new transaction for processing each parallel rule. The initiated transaction ends with an enqueue to the Mediator parallel message dehydration store.

  For example, if a Mediator service component has one parallel routing rule, one message is enqueued on the 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.

- Mediator commits or rolls back transactions because it is the initiator of these transactions.

Finer Control Over Thread Allocation in Parallel Routing

You can specify dedicated work managers to handle parallel routing and error handling messages for a mediator component. You can use the Oracle WebLogic Server Administration Console to configure work managers. See "Viewing and Configuring Work Manager Properties" in *Administering Oracle SOA Suite and Oracle Business Process Management Suite* for more details on configuring work managers.

Use the `NameWorkManagerMappings` Mediator service engine property to specify the mediator component and its associated work managers in Oracle Enterprise Manager Fusion Middleware Control. See "Configuring Oracle Mediator Service Engine Properties" in *Administering Oracle SOA Suite and Oracle Business Process Management Suite* for more details on configuring Mediator runtime properties.

The **NameWorkManagerMappings** property has the following keys:

- **parallelRoutingWorkManagerName:** The name of the work manager configured for parallel routing. If this is not specified, the default SOA work manager is used.
- **fullyQualifiedComponentDistinguishedName:** The fully qualified distinguished name of the mediator component. The format to be used is *`PartitionName/CompositeName`*`!`*`Revision/ComponentName`*`.` For example, `soaInfra/MyProject!1.0/Mediator1.`

Resequencing Rules

Mediator includes a resequencer, which rearranges streams of related but out-of-sequence messages into their sequential order based on the type of resequencer used and the rules you define. When incoming messages arrive in a random order, the resequencer orders the messages based on sequential or chronological information, and then sends the messages to the target services in the correct order based on the resequencing configuration.

For more information about resequencing messages, see Resequencing in .

Defining Routing Rules

Routing rules can only be defined for a Mediator with a defined interface. For more information on how to define an interface, see How to Define an Interface for a Mediator .

How To Access the Routing Rules Section

You define the routing rules in the **Routing Rules** section of the Mediator Editor.

Figure 20-1 shows the **Routing Rules** section of the Mediator Editor.

Figure 20-1 Mediator Editor- Routing Rules Section

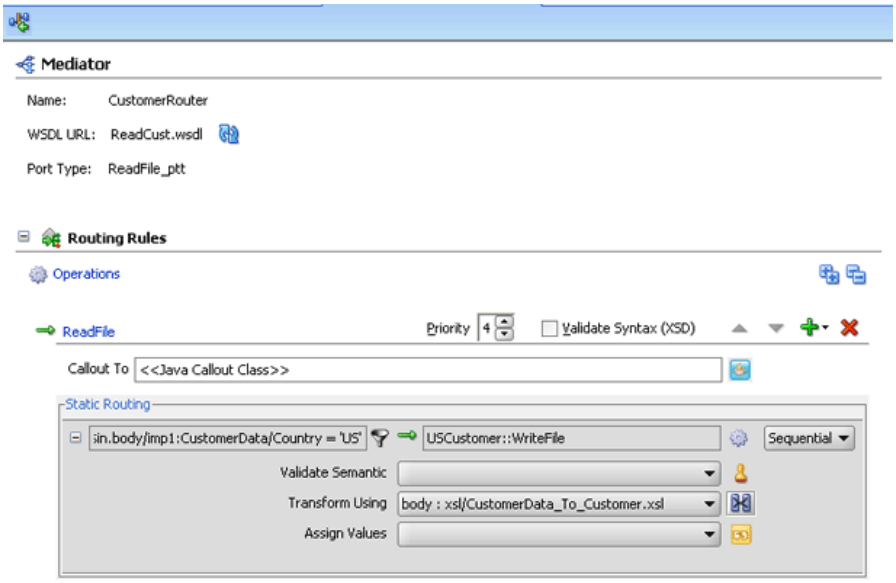


Figure 20-2 lists and describes the icons in the **Routing Rules** section.

Figure 20-2 Routing Rule Section Icons

You can access the **Routing Rules** section of the Mediator Editor using one of the following methods:

From the SOA Composite Editor:

1. Double-click the icon that represents the Mediator for which you want to specify the routing rules.
2. If the **Routing Rules** section is not visible, click the **Plus** (+) icon next to **Routing Rules**.

From the Applications window:

1. In the Applications window, expand the SOA project and then expand the **SOA Content** folder.
2. In the **SOA Content** folder, double-click the name of the Mediator file in which you want to specify the routing rules.

   The Mediator file has an `MPLAN` extension.
3. If the **Routing Rules** section is not visible, click the **Plus** (+) icon next to **Routing Rules**.

How to Create Static Routing Rules

The following topics provide information and instructions for defining static routing rules for Mediator, including specifying the services and events, defining handlers, transformations, expressions, filters, and so on.

How to Specify Mediator Services or Events

After creating a 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 a 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 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.

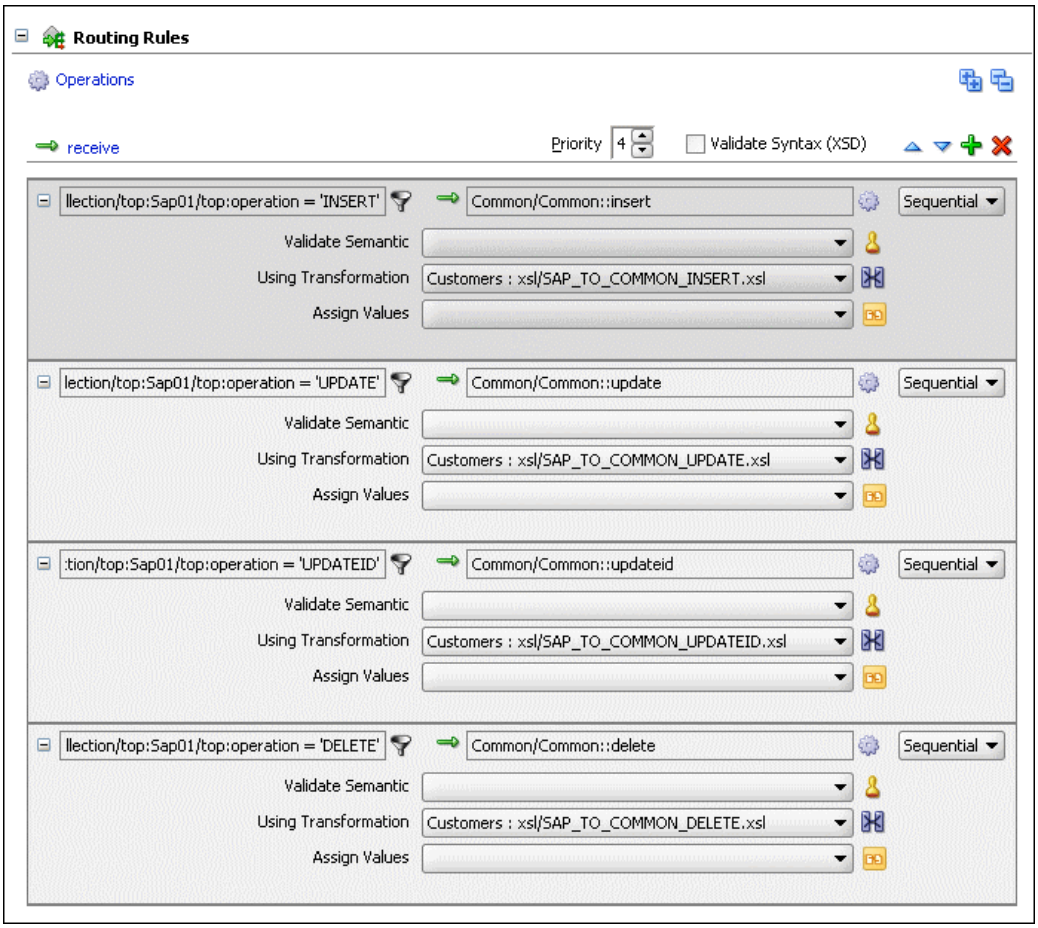
The purpose of the echo option is to expose all the Mediator functionality as a callable service without having to route it to any other service. For example, you can call a Mediator to perform a transformation, a validation, or an assignment, and then echo the Mediator back to your application without routing 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 20-3.

Figure 20-3 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 20-4.

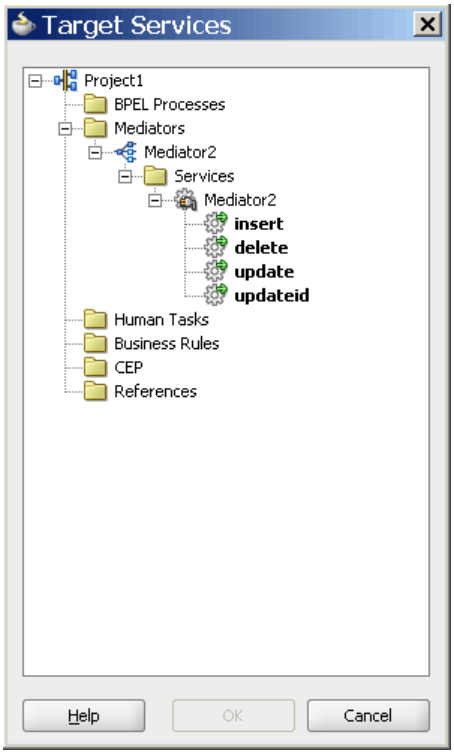
   Figure 20-4 Target Type Dialog

2. Click **Service**.

   The Target Services dialog appears, as shown in Figure 20-5.

Figure 20-5 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 20-5.

---

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 20-4.

2. Click **Event**.

   The Event Chooser dialog appears.

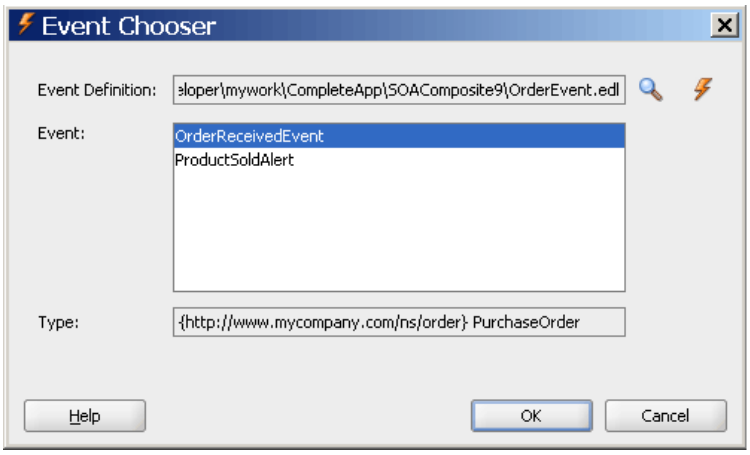
3. To the right of the **Event Definition** field, click **Search**.

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 20-6.

Figure 20-6 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 20-7.

Figure 20-7 Target Type Dialog

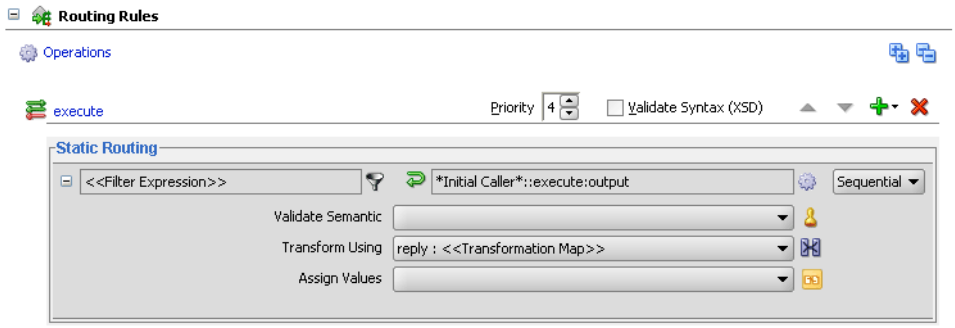
2. Click **Echo**.

Note:

The Echo button only appears on the Target Type dialog if the interface is synchronous or asynchronous.

Figure 20-8 shows a routing rule with a synchronous echo. An asynchronous echo has an icon with a dotted line on the return.

Figure 20-8 Sample Mediator Supporting Echo Operation



What You May Need to Know About Echoing a Service

The echo option has the following limitations:

- Echoing a service is supported only with Mediator interfaces having the following types of WSDL files:
  - Request/reply
  - Request/reply/fault
  - Request/callback

  Note:

  The echo option is not available for Mediator interfaces having request/reply/fault/callback WSDL files or for one-way WSDL files.

- The echo option is available for synchronous operations such as 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 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 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 Mediators with asynchronous operations.

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.

How to Configure Response Messages

In the 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 a 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 Mediator between the first Mediator and the two-way service. The response should first be forwarded to the one-way Mediator, which in turn should call the two-way service.

Note:

- Zero is an unsupported value to be specified as a timeout period.
- If the callback is received 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 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 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.

How to Handle Premature Callbacks

Callback messages might arrive before the initiating transaction is completed. In this case, correlation in Mediator fails. If you have an issue with premature callbacks, you can use the `oracle.tip.mediator.callback.correlationWaitDuratino_in_seconds` property to set a time period in seconds for which the callback thread waits before retrying the callback.

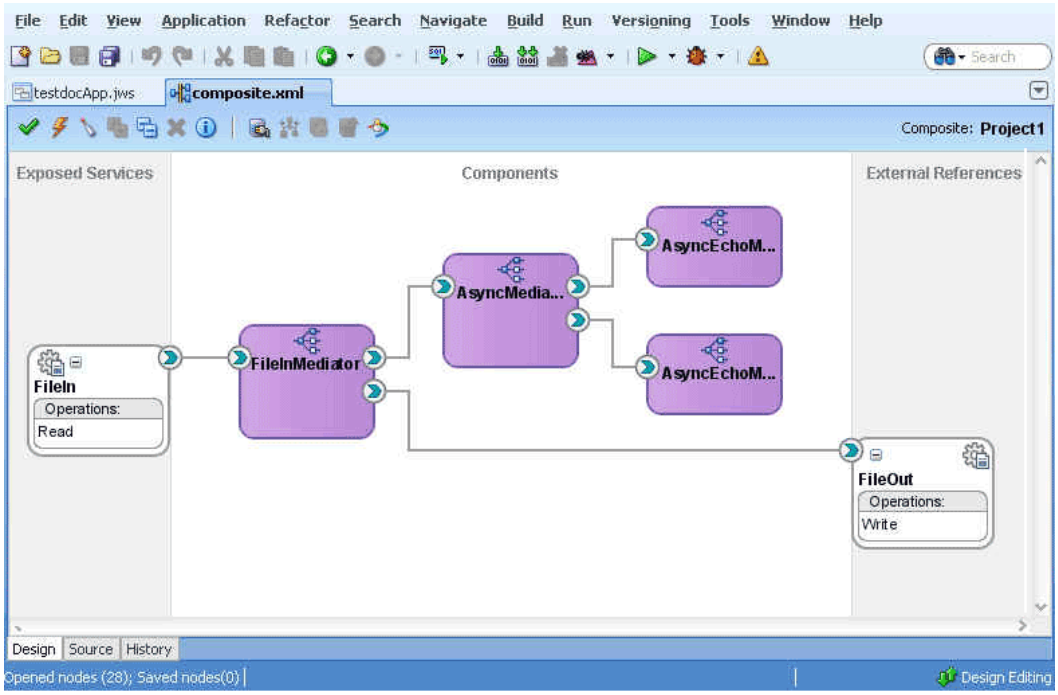
You define the property in the `composite.xml` file in the `component` element that defines the Mediator component. In the example shown below, the wait time before retrying is 15 seconds.

```
<component name="Mediator1">
    <implementation.mediator src="Mediator1.mplan"/>
    <property name="oracle.tip.mediator.callback.correlationWaitDuration_in_
    seconds">15</property>
</component>
```

How to Handle Multiple Callbacks

A single Mediator cannot handle multiple callbacks. If you have a composite application with a Mediator that receives multiple callbacks, the behavior of the composite application is undetermined. For example, in the scenario shown in Figure 20-9, **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 20-9 Sample Mediator Handling Multiple Callback



How to Handle Faults

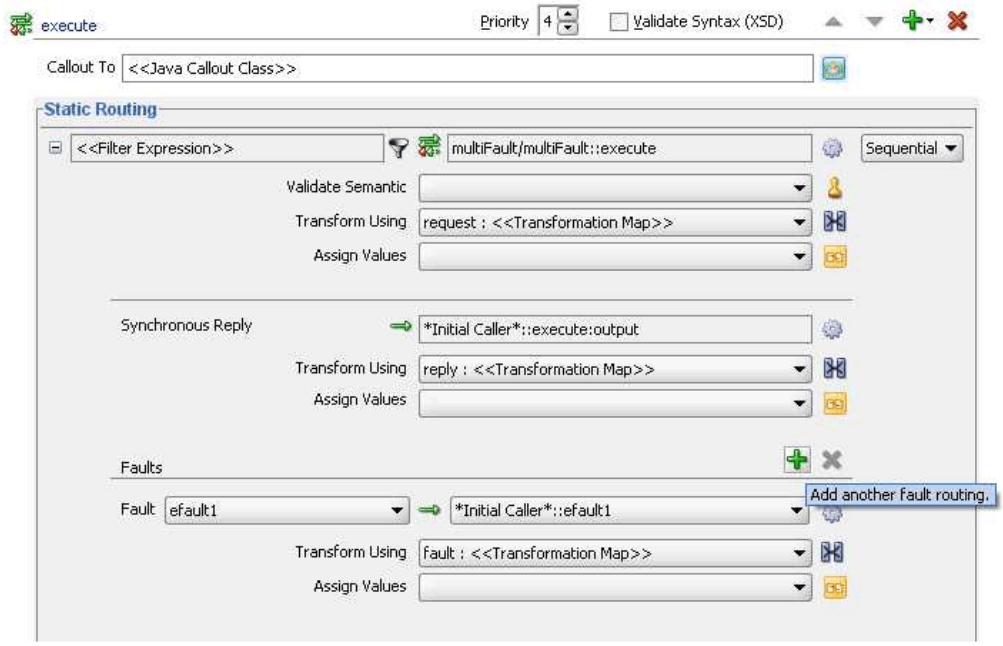
If you create a new routing rule in which the target service operation has one or more faults, you still see a single fault routing section in the Mediator Editor. If the source Mediator service component supports one or more faults, then the fault is routed back to the caller by default. You can choose 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 20-10.

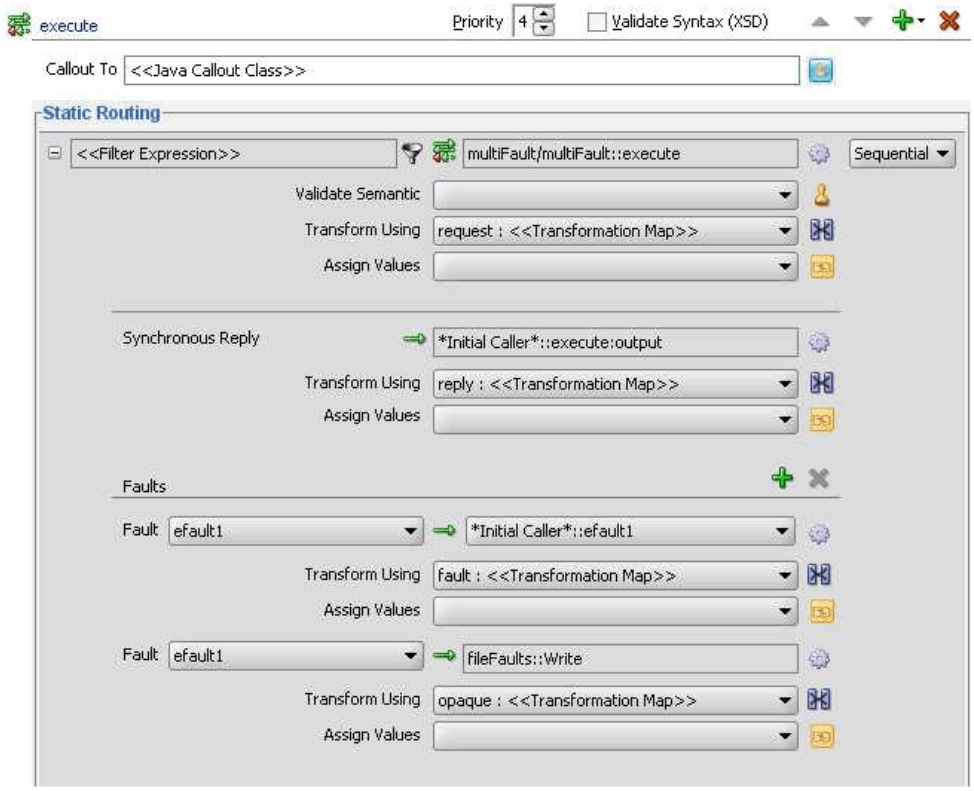
Figure 20-10 Adding a Second Fault



Another fault section appears in the routing rule box.

2. Configure the target service, transformations, and assign values for the new fault. Figure 20-11 shows a second fault being routed to a file adapter service.

Figure 20-11 Second Fault Added to Routing Rules



Note:

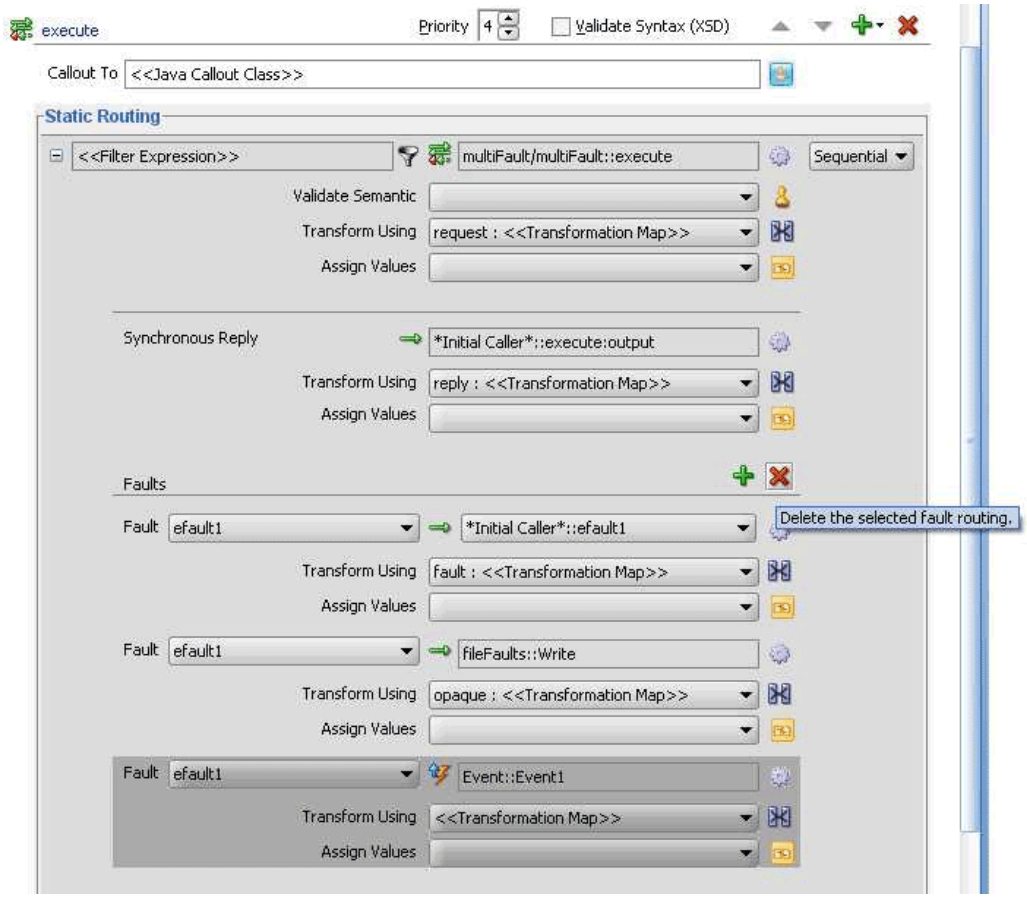
You can route the same fault to multiple targets using different transformations.

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 20-12.

Figure 20-12 Deleting a Fault Routing



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 below:

```
$in.property.custom.Priority = '1'

$in.property.tracking.ecid = '2'
```

Filter Expression Message Headers

Two examples of filter expression message headers are shown below:

```
$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 the following example to the root element of the `.mplan` file.

```
wsse = "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
 secext-1.0.xsd"
```

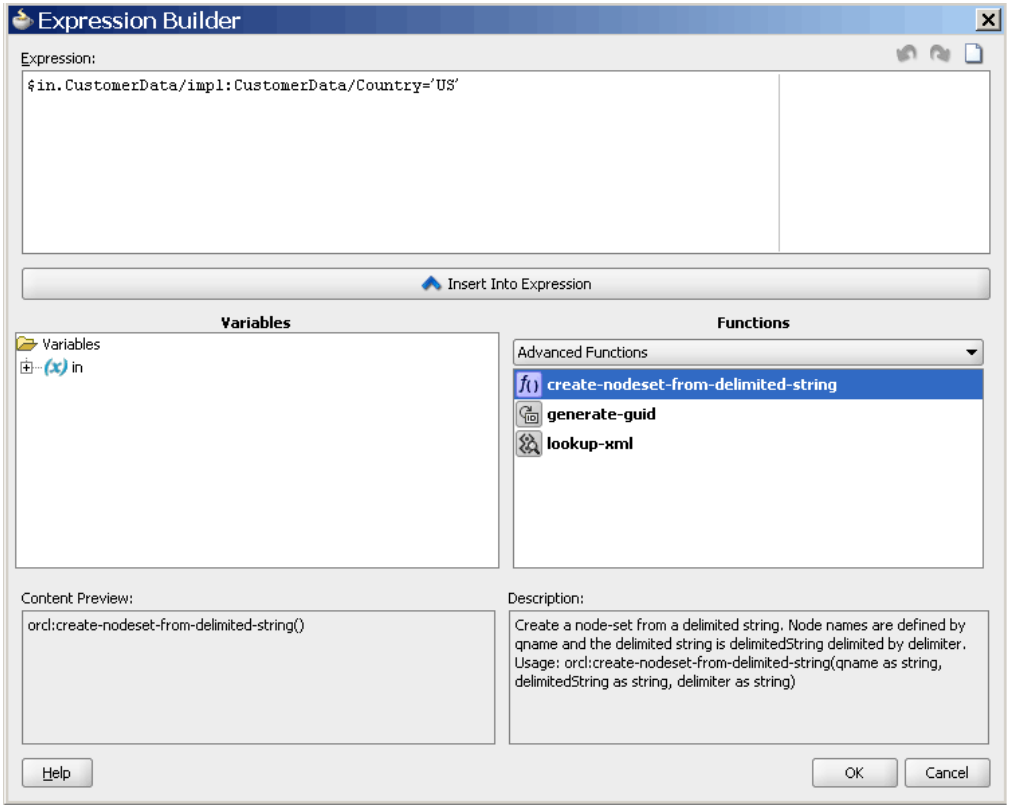
To specify an expression for filtering messages:

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 20-13.

Figure 20-13 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 table describes each of the fields in the Expression Builder dialog:

Table 20-1 Expression Builder Fields

Field	Description
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 a Mediator component. Oracle JDeveloper parses the Mediator WSDL file and presents the message definition in the **Variables** field. The input message is stored in the `$in` variable, and you can use the `$in.properties` to access the properties of an input message. If the input message consists of multiple parts, use `$in.`*`partname`* to access a part of an input message.

Table 20-1 (Cont.) Expression Builder Fields

Field	Description
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	This field describes the value selected from the **Variables** field or **Functions Palette**.

To specify a filter expression on a message payload:

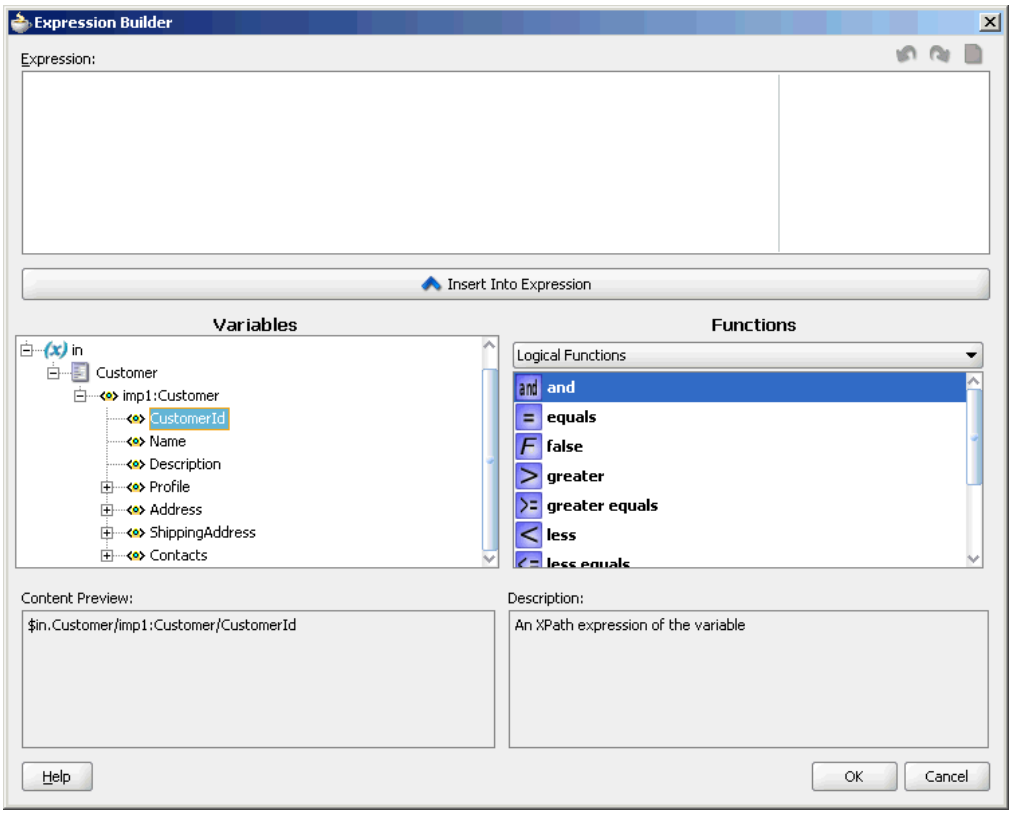
1. To the right of the **Filter Expression** field in the **Routing Rules** section, click the **Invoke Expression Builder** icon.

   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 20-14.

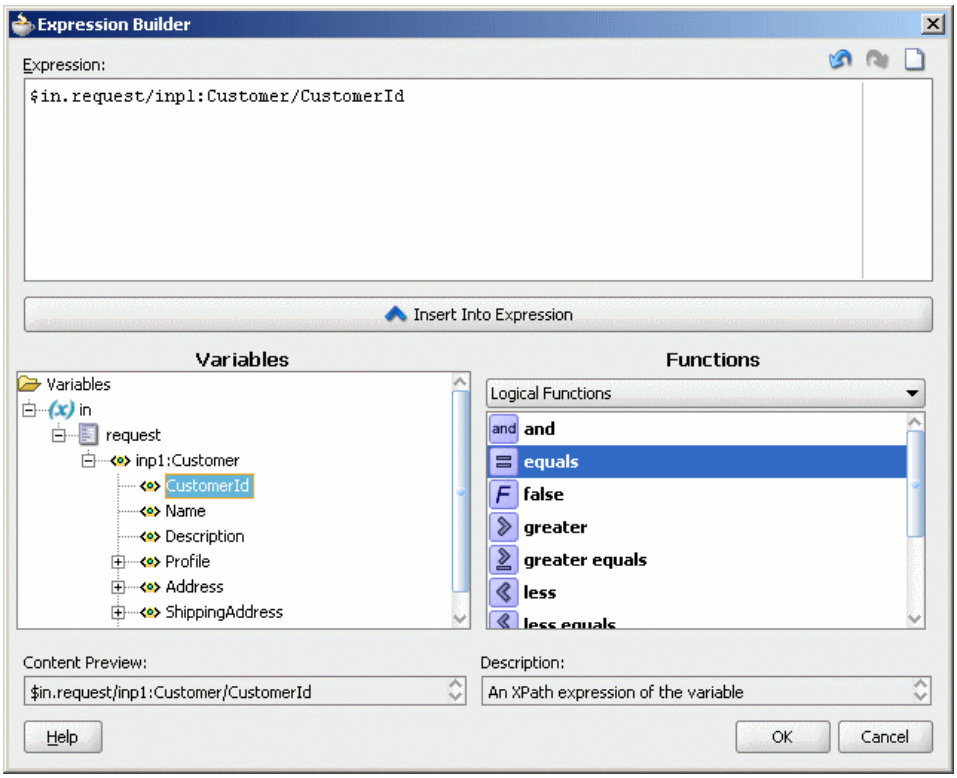
Figure 20-14 Expression Builder Dialog – Variables Element Selected



3. Click **Insert Into Expression.**

The expression is added in the **Expression** field, as shown in Figure 20-15.

Figure 20-15 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 20-15.

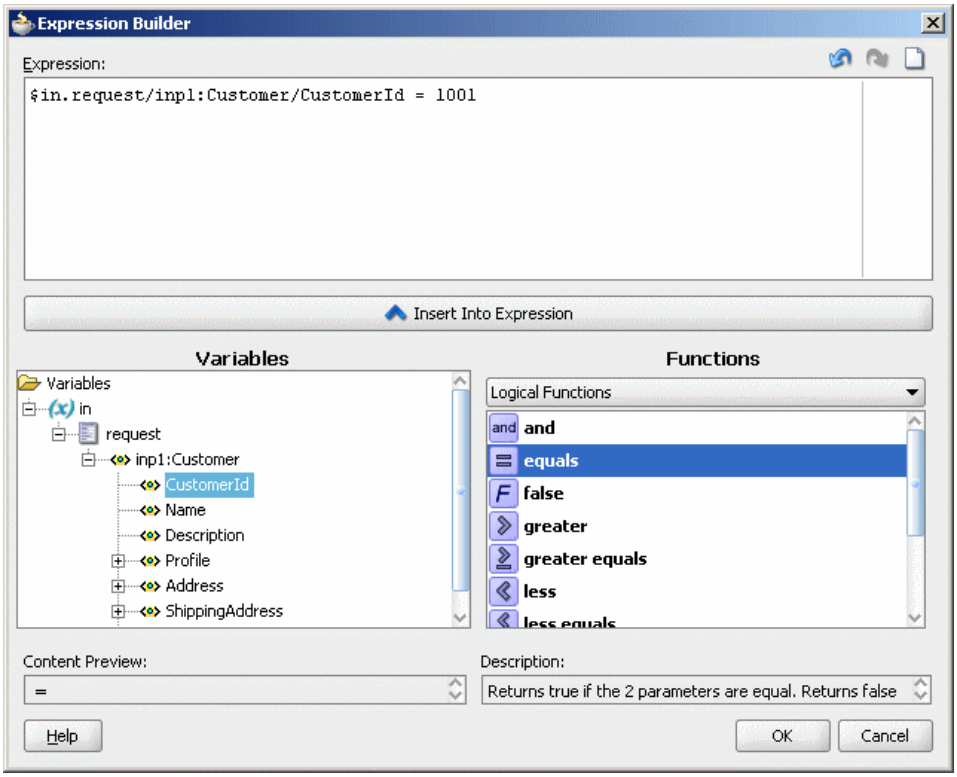
5. Click **Insert Into Expression**.

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

6. Complete the expression.

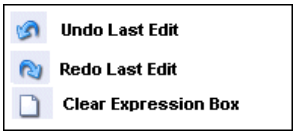
   In this example, the Customer ID must equal`1001` to evaluate to true, as shown in Figure 20-16.

Figure 20-16 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 20-17.

Figure 20-17 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.

How to Translate Between Native XSD Formats and XML Formats

Mediator enables you to translate native format data into XML data, for inbound data, and XML data into native format data for outbound translations. So, for example, you can use inbound translation to convert an incoming comma-delimited native data file into an XML data file. You can use outbound translation to convert XML data into native data format for a target service.

Mediator provides the following translation features:

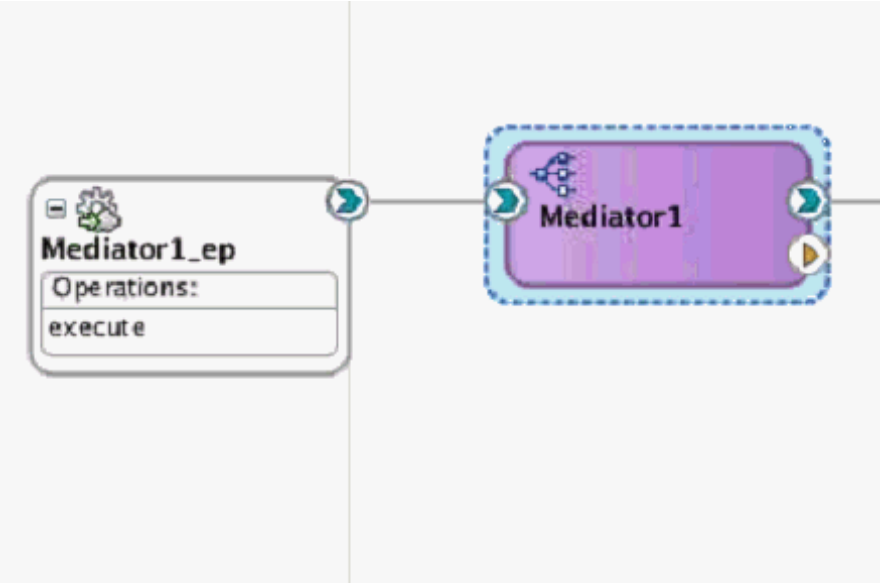
- Inbound Translation: Converts inbound data from native data format to XML. Inbound translation is configured at the operation level. The translated data is available for transform and assign operations.

- Outbound Translation: Converts outbound data from XML to native data format. Outbound translation can be configured for each routing rule. The native data is then routed to the target service.
- Translate and Route Only: Translates inbound data from native data format to XML, and routes it to the target service. An outbound WSDL file is created for the target service. This feature is only supported for mediators that have a one-way (no response) operation.

How to Use Inbound Translation

This section demonstrates using inbound translation. Figure 20-18 shows a mediator (Mediator1) connected to an inbound web service. The mediator receives a native string from the inbound web service, and uses inbound translation to convert the native string into XML.

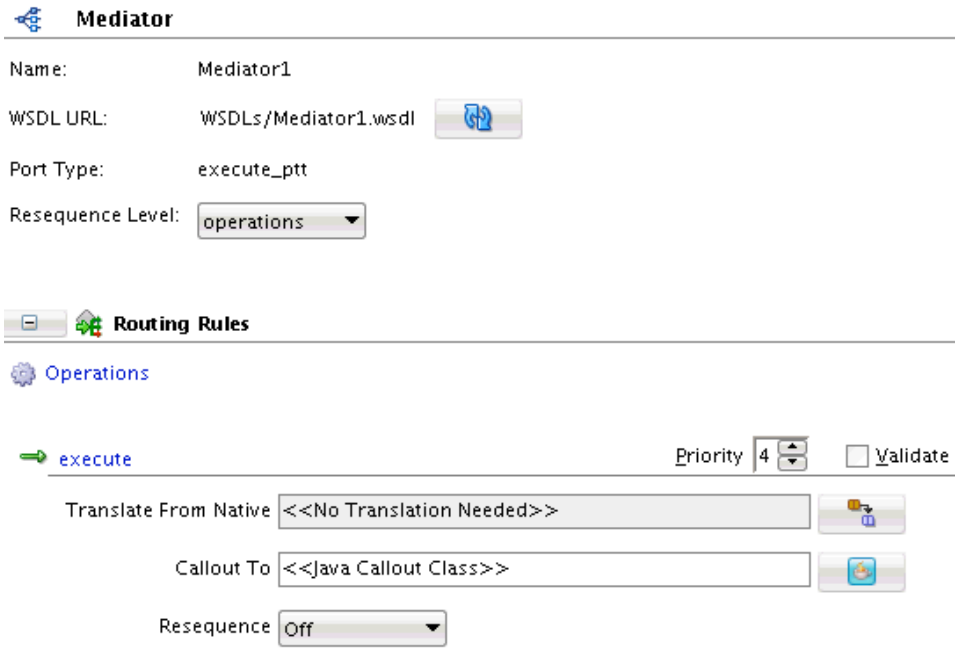
Figure 20-18 Mediator Receiving Inbound Data



To translate inbound data from native XSD to XML format:

1. Right-click the mediator (Mediator1), and select **Edit**.
2. Under the Operations section, click the icon to the right of the **Translate From Native** field. Figure 20-19 shows the Operations section for Mediator1.

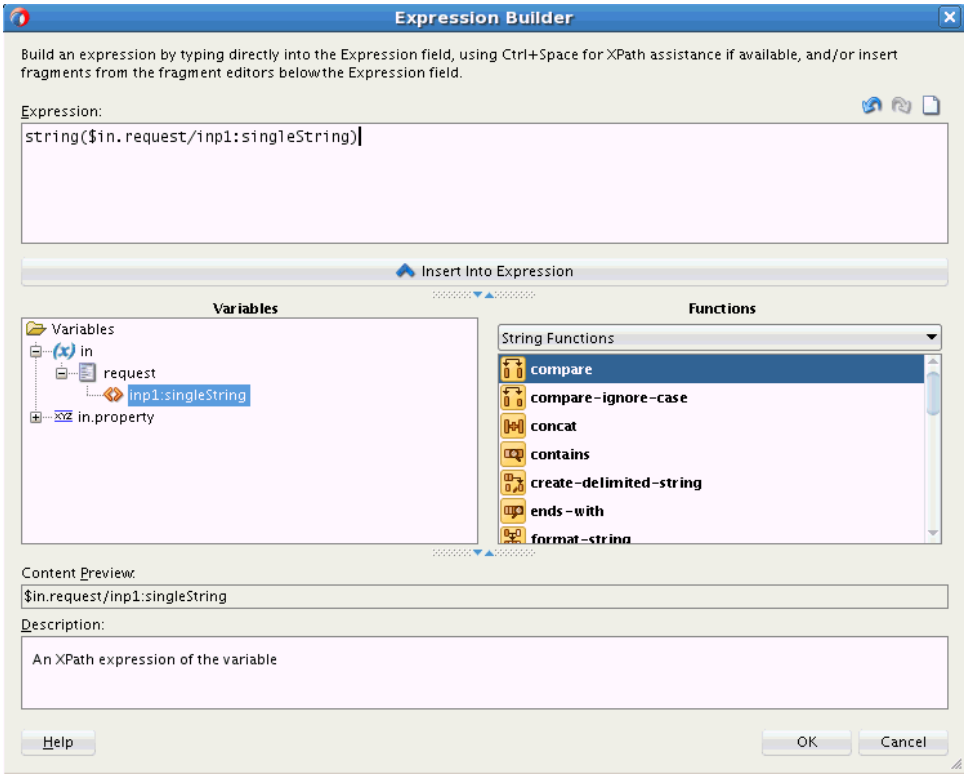
Figure 20-19 Translate From Native Option



3. In the Add Translation dialog box that appears, click the icon to the right of the Input field. The Expression Builder dialog appears.

4. Double-click the input string in the Variables tree. Wrap (cast) the input string that appears in the Expression field with the `string()` function. Figure 20-20 shows the Expression Builder dialog with the completed input string. Click **OK**.

Figure 20-20 Completed Input String in Expression Builder

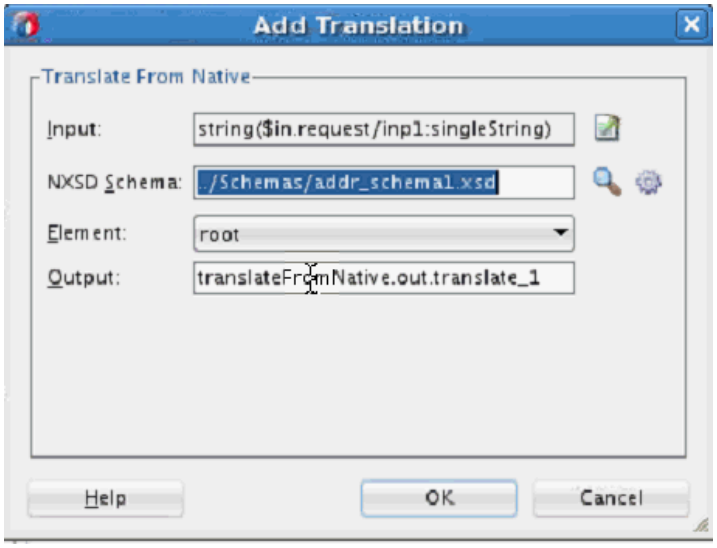


5. To the right of the **NXSD Schema** field, select the **Search** icon to invoke the Type Chooser dialog for selecting the schema. If the schema does not exist, you can click the second icon to create the schema.

6. Select the schema, and click **OK**. The **Element** field is populated from the selected schema. The **Output** field is populated with an intermediate output variable created by Mediator. This variable must be in the format `translateFromNative.out.`*`some_name`*`.`

   Figure 20-21 shows the completed Add Translation dialog.

Figure 20-21 Add Translation Dialog



7. Click **OK**. The Translate From Native field is populated.

How to Use Outbound Translation

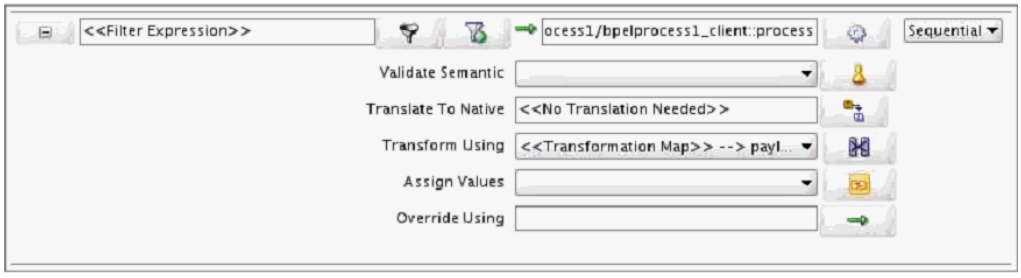
This section demonstrates using outbound translation. Figure 20-22 shows a mediator (Mediator1) connected to a BPEL process. The mediator uses outbound translation to convert XML data into native string, and routes this string to the BPEL process.

Figure 20-22 Mediator Sending Outbound Data

To translate outbound data from XML to native XSD format:

1. Right-click the mediator (Mediator1), and select **Edit**.
2. Under the routing rule that routes data from the mediator to the BPEL process (target service), click the icon to the right of the **Translate To Native** field. Figure 20-23 shows the routing rule section for Mediator1.

Figure 20-23 Translate To Native Option

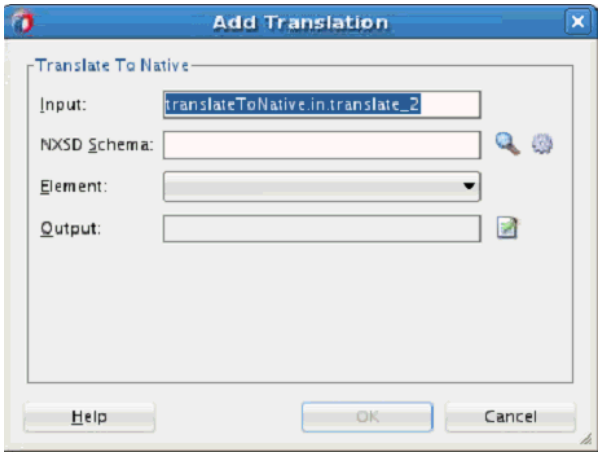


3. In the Add Translation dialog box that appears, optionally edit the default input variable in the **Input** field. The **Input** field is populated with an intermediate input variable created by Mediator. This variable must be in the format `translateToNative.in.`*`some_name`*`.`

> **Note:**
>
> You can later assign a value to the intermediate input variable using the Assign or Transform action of the associated routing rule.

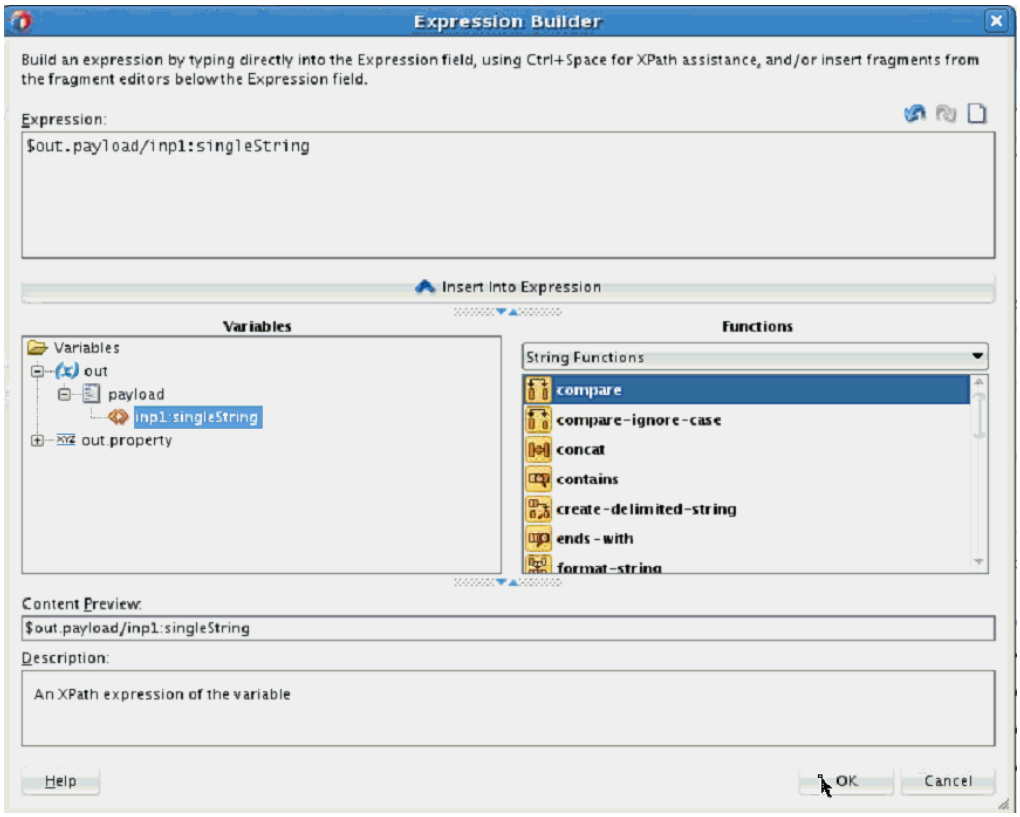
Figure 20-24 shows the Add Translation dialog box.

Figure 20-24 Add Translation Dialog



4. To the right of the **NXSD Schema** field, select the **Search** icon to invoke the Type Chooser dialog for selecting the schema. If the schema does not exist, you can click the second icon to create the schema.
5. Select the schema, and click **OK**. The **Element** field is populated from the selected schema.
6. Click the icon to the right of the **Output** field. The Expression Builder dialog appears.
7. Double-click the output string in the Variables tree. Click **OK**.

Figure 20-25 Completed Output String in Expression Builder



8. Click **OK** in the Add Translation dialog box.

How to Create XSLT Transformations

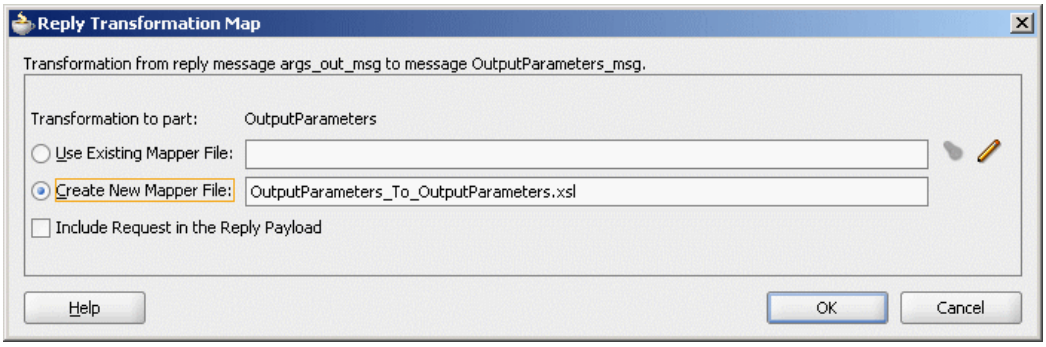
Oracle JDeveloper provides an XSLT Mapper that lets you specify a mapper file (XSL file) to transform data from one XML schema (expressed as an XSD file) to another. The XSLT Mapper enables data interchange among applications using different schemas. For example, you can map an incoming purchase order 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.
2. Do one of the following:
   - If the XSLT map file (.xsl) exists, click **Browse** to find and select the XSLT file to use. Click **OK.**
   - If you are creating a new XSLT map file, click the **Create Mapping** icon. The Create Transformation Map dialog appears.
3. In the Create Transformation Map dialog, select XSLT under **Type.**
4. Edit the XSLT **File Name,** as appropriate.
5. Edit the XSLT **Directory,** as appropriate. The default directory is the *SOA_Project*`/SOA/Transformations` directory. Click **Browse** to browse and select the directory.
6. 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 20-26.

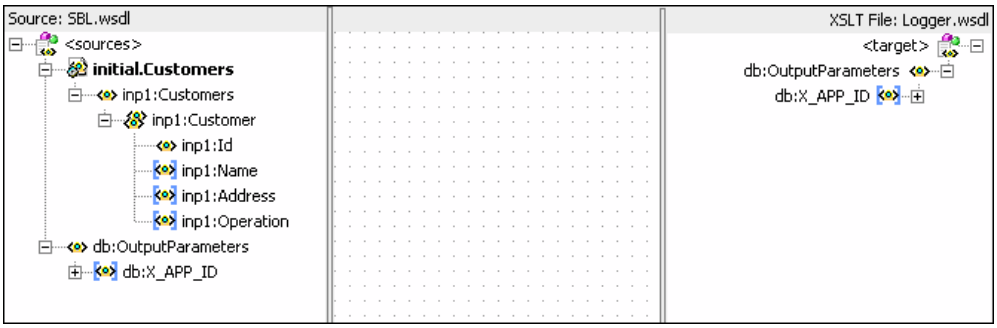
Figure 20-26 Reply Transformation Map Dialog



7. 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 20-27.

Figure 20-27 Initial Variable in XSL File



Note:

An initial message can also consist of multiple parts. 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 Creating Transformations with the XSLT Map Editor .

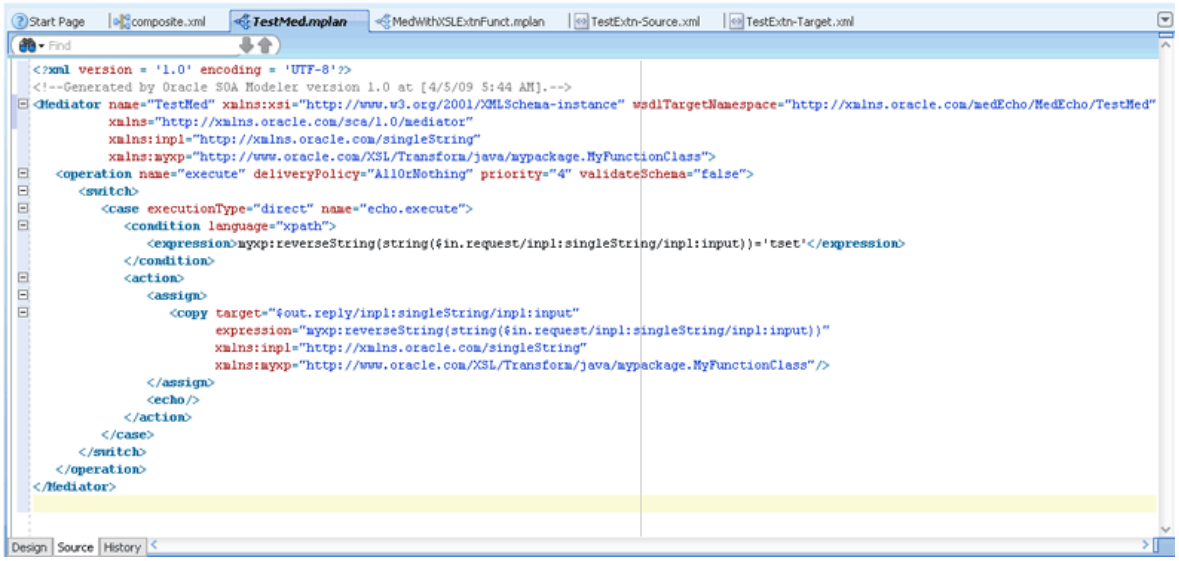
To add user-defined extension functions:

You can use the Expression Builder to include user-defined extension functions.

1. Create an XPath function.
2. Register the Jaxen XPath function with a 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 20-28.

Figure 20-28 Project .mplan file – Modified to Use User-Defined Extension Functions



8. Invoke the test page with a suitable payload.

How to Create XQuery Transformations

Oracle Mediator supports XQuery transformations from one XML schema to another. The XQuery 1.0 specification is supported.

To create an XQuery 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.

2. Do one of the following:

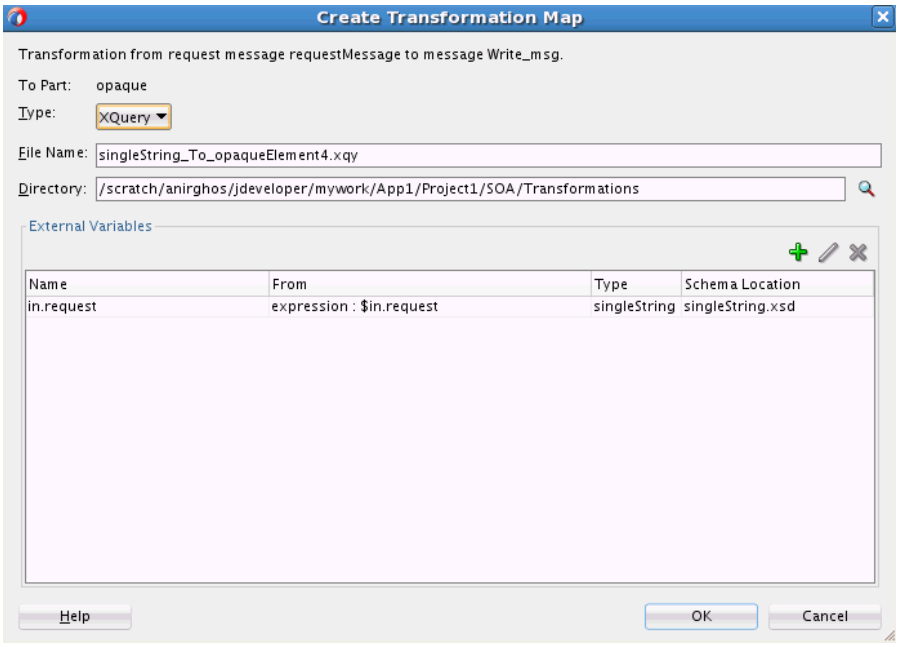
   - If the XQuery map file (.xqy) exists, click **Browse** to find and select the XQuery file to use. Click **OK.**

   - If you are creating a new XQuery map file, click the **Create Mapping** icon. The Create Transformation Map dialog appears.

3. In the Create Transformation Map dialog, select XQuery under **Type.**

   Figure 20-29 shows the Create Transformation Map dialog.

Figure 20-29 Create Transformation Map Dialog



4. Edit the XQuery **File Name,** as appropriate.

5. Edit the XQuery **Directory,** as appropriate. The default directory is the *SOA_Project*`/SOA/Transformations` directory. Click **Browse** to browse and select the directory.

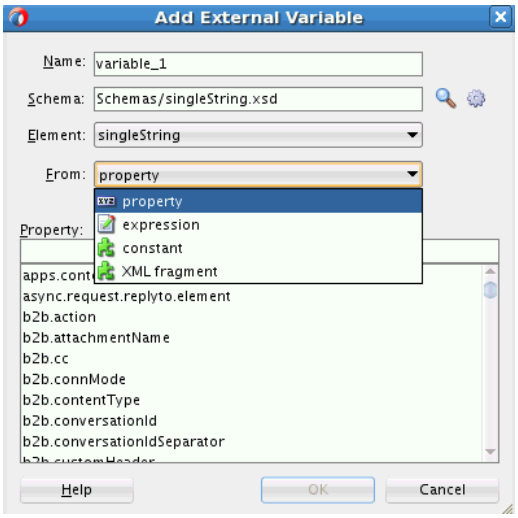
6. Under the External Variables section, you can define the external variables for the XQuery. Click **Add Variable** to add a new external variable. The Add External Variable dialog appears.

Note:

External variables are automatically created for implicit mediator variables that are available as transformation input. For example, the mediator input request `in.request` automatically has an external variable for it.

Figure 20-30 shows the Add External Variable dialog.

Figure 20-30 Add External Variable Dialog



7. Specify a **Name, Schema,** and schema **Element** for the external variable.

8. Under **From,** choose how to map the value for the external variable. Select from one of the following:

   - Property: Lists the properties that you can select from.
   - Expression: Enables you to build an expression using mediator implicit variables, properties, and a list of functions that you can use in the expression. You can click the **Invoke Expression Builder** icon to launch the expression builder.

     See How to Specify an Expression for Filtering Messages and Building XPath Expressions in the Expression Builder in for more information about working with the expression builder.
   - Constant: Enables you to specify a literal value for the external variable.
   - XML Fragment: Enables you to specify XML data for the external variable.

9. Click **OK** in the Add External Variable dialog to add the external variable. The Create Transformation Map dialog is populated with the external variable.

   Note:

   To edit an external variable, select it from the list and click **Update Variable.**

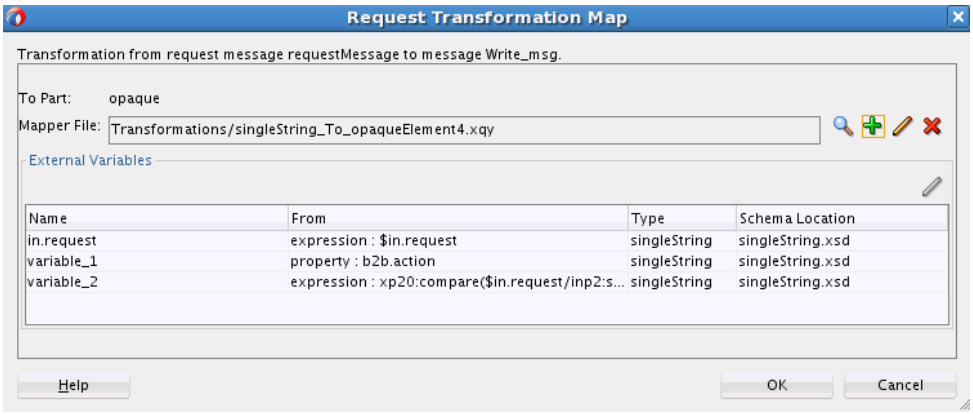
   To delete an external variable, select it from the list and click **Delete Variable.**

   ---

10. Click **OK** in the Create Transformation Map dialog. The Request Transformation Map dialog appears, and it is populated with the **Mapper File** name and the external variables defined.

    Figure 20-31 shows the Request Transformation Map dialog.

Figure 20-31 Request Transformation Map Dialog



11. Click **OK** in the Request Transformation Map dialog. The transformation action details are added to the mediator mplan file.

To edit an XQuery 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.

   Note:

   You cannot add or delete external variables from an existing XQuery (.xqy) map. However, you can select a variable and click **Update Variable** to modify the expression or value associated with the external variable.

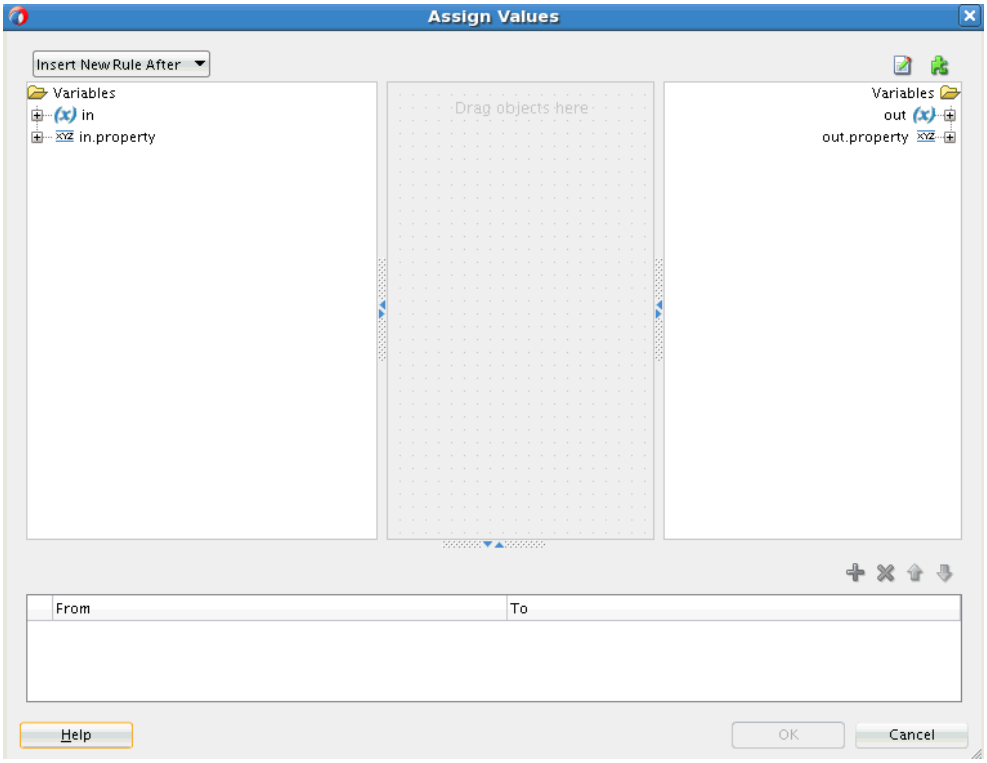
2. Click the **Edit Mapping** icon to the right of the Mapper File field. The XQuery map opens in the XQuery Mapper.

3. See Creating Transformations with the XQuery Mapper for more information on using the XQuery Mapper.

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 20-32 shows the Assign Values dialog that is displayed when you click the **Assign Values** icon in the **Routing Rules** section.

Figure 20-32 Assign Values Dialog



The left hand pane of the Assign Values dialog contains the source variables and the right hand pane shows the target variables. You can copy values from source variables to target variables. You can also create complex expressions and assign them to target variables. You can also assign literals (constants or XML fragments) to target variables.

The bottom pane of the Assign Values dialog shows the assignments you have created. You can select and edit any assignment.

To copy a source node to a target node:

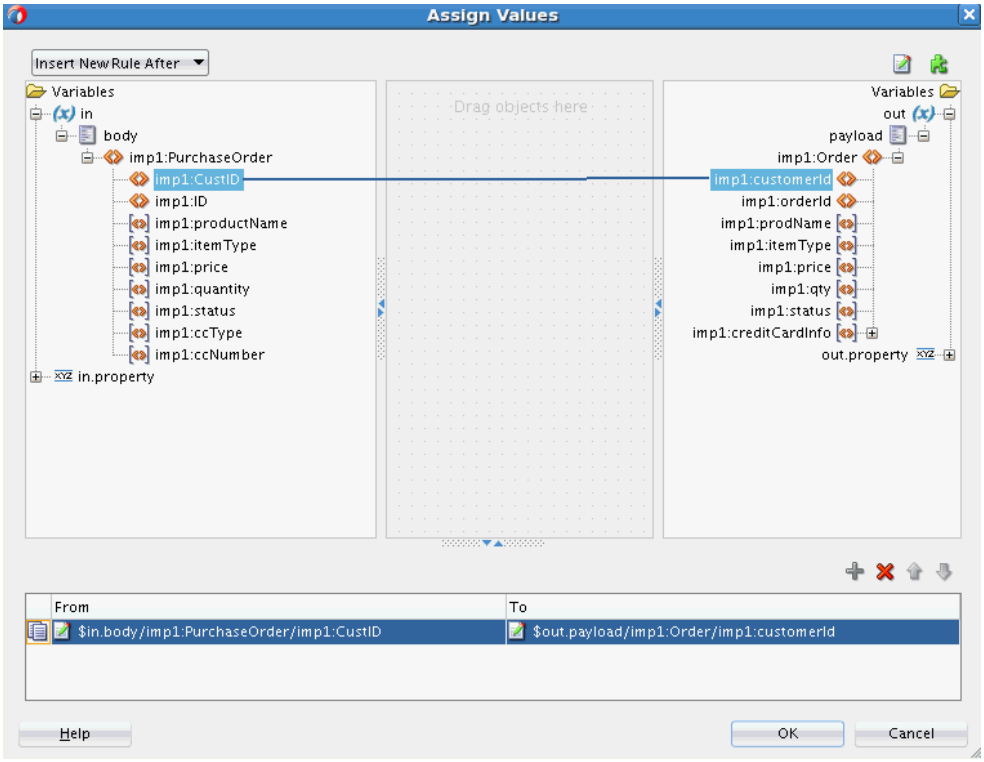
1. Expand the source tree in the left pane by clicking the plus sign (+) next to a source node. Similarly expand the target tree in the right pane.

2. Use one of the following methods to copy a source variable to a target variable:

   - Drag the desired source node to the target node. A line appears connecting the source and target nodes. The assignment also appears in the bottom pane. Figure 20-33 shows the Assign Values dialog after copying a source node to a target node.

   - Select the source node in the left pane and the target node in the right pane. Click the **Create rule from selected nodes** icon (green plus icon) above the bottom pane to create an assignment.

Figure 20-33 Copying Source Variables to Target Variables



3. Click **OK** to create the assignment.

To assign complex expressions:

1. Drag the **Assign Source Expression** icon from the top right hand corner to the target node or the canvas (center pane). The Expression Builder appears.
2. Create an expression using the available source variables and functions.

   Optionally click **Help** for more information about the Expression Builder dialog.
3. Click **OK** to close the Expression Builder.
4. If you had dragged the expression to the canvas or center pane in Step 1, drag the expression icon in the canvas to the desired target node. This maps the expression to the target variable.

Note:

To edit the assignment, right-click the assignment in the bottom pane. Select **Edit Source Expression** or **Edit Target Expression** to edit the source and target respectively.

---

5. Click **OK** to create the assignment.

To assign constant values and XML fragments:

1. Drag the **Assign Source Literal** icon from the top right hand corner to the target node or the canvas (center pane). The Assign Source Literal dialog appears.

2. Enter the constant value or XML Fragment to be assigned.
3. Select **Literal is XML Fragment** if your constant value is valid XML.
4. Click **OK**.
5. If you had dragged the source literal to the canvas or center pane in Step 1, drag the source literal icon in the canvas to the desired target node. This maps the source literal to the target variable.

Note:

- When you assign values to a particular 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 20-2 through Table 20-4 list the various possibilities of assignment on constants and properties, payloads, and headers of a message from source to target.

Table 20-2 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 20-3 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"/>`

Table 20-3 (Cont.) Possibilities on Payload

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

Table 20-4 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"/>`

Table 20-4 (Cont.) Possibilities on Header

Source	Target	Example
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"/>`

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 of the `<copy>` elements that appear in the Mediator mplan.
- You can reorder the assignments by selecting an assignment in the bottom pane of the Assign Values dialog and clicking the Up or Down arrow to move the assignment in the assignments list.
- When creating a new assignment, you can choose to insert it at the desired place in the list of assignments. Select an existing assignment in the bottom pane of the Assign Values dialog and select **Insert New Rule After** or **Insert New Rule Before** from the list at the top left of the dialog.
- The output variable from the Translate From Native activity and the input variable to a Translate To Native activity are also available for assignments in the Assign Values dialog.
- All assignments that appear in the bottom pane of the Assign Values dialog are applied to the Mediator mplan only after you click **OK.**
- 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. The following example provides details:

```
<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. The following example provides details.

```
<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. The following example provides details.

```
<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. The following example provides details:

  ```
  <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. The following example provides details:

  ```
  <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. The following example provides details:

```
<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. The following example provides details:

```
<ProductReq>
        <Make>MAKE</Make>
        <Model>MODEL</Model>
</ProductReq>
```

- If there are multiple assign activities in a 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. The following example provides details.

```
<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 the preceding example, 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.

How to Access Headers for Filters and Assignments

When the Expression Builder is invoked from a Mediator, either for defining a filter or for defining an assignment source or target, the WSDL file is parsed. This automatically detects any SOAP headers for the current routing rule operation and makes them visible as variables under the `in` or `out` folder as `header./ns_elementName/`, as shown in Figure 20-34. Here, `ns` is the namespace prefix and `elementName` is the root element name for the header schema.

The following scenarios provide details.

Scenario 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
```

Scenario 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.

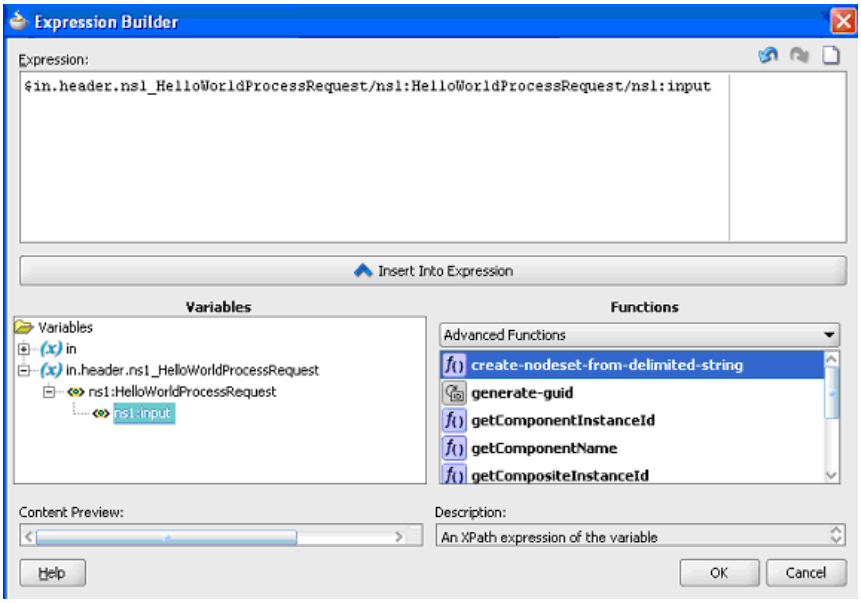
For example, enter the expression in the Expression Builder shown in the following example:

```
$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 the following example:

```
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 20-34 Expression Builder Dialog - Automatic Header Detection



By default, SOAP headers are not passed through by Mediator. You must add the `passThroughHeader` endpoint property to the corresponding Mediator routing service:

```
<property name="passThroughHeader">true</property>
```

For example, to add this property, you can modify the `composite.xml` file, as shown in the following example:

```
<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` Mediator (without a transformation or assign), if the source and target part QNames are not identical, then 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.

Note:

- 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 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.

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 a 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 the following example:

```
$in.header.<header root element namespace prefix>_<header root element name>/<xpath>
```

Therefore, for the header shown in the following example:

```
<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 the following example:

```
<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 the following example to the root element of the `.mplan` file.

```
wsse = "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd"
```

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"/>
```

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.

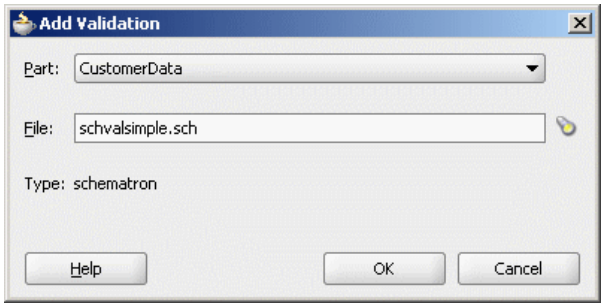
5. Select a Schematron file and click **OK**.

   Note:

   - Schematron files usually have a `.sch` extension.
   - No error message or warning is displayed if the selected Schematron file is empty.

   The Add Validation dialog is updated, as shown in Figure 20-35.

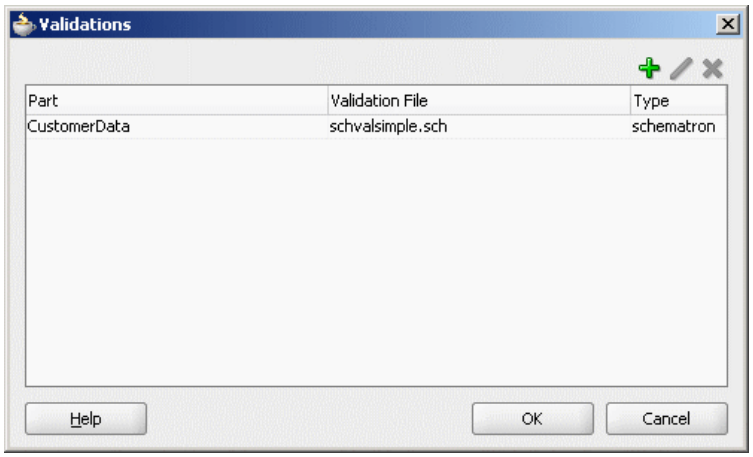
Figure 20-35 Add Validation Dialog



6. Click **OK**.

   The Validation dialog is updated, as shown in Figure 20-36.

Figure 20-36 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.

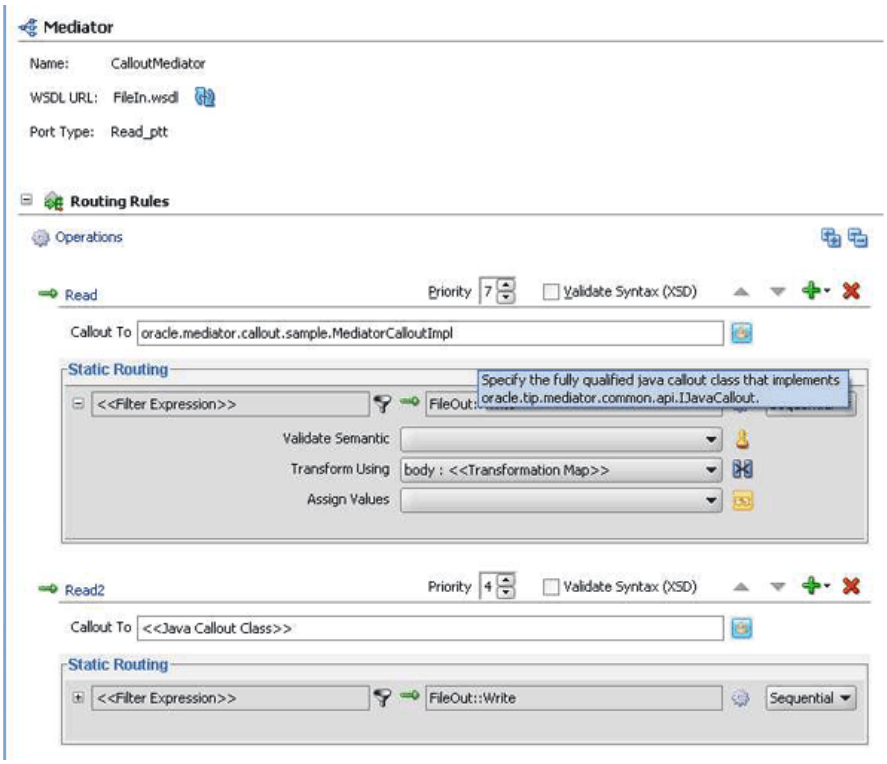
How to Work with Attachments

You can configure how Mediator handles attachments by adding properties to the project's `composite.xml` file. For information on working with attachments, see "Sending Attachment Streams" and "Overriding Pass Through Settings for Attachments in ".

How to Use Java Callouts

Java callouts enable you to use external Java classes to manipulate messages flowing through the Mediator. Only one Java callout is supported per 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 20-37 shows a sample Mediator with two operations, in which both the operations have one routing rule each and the first operation has a callout class.

Figure 20-37 Sample 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.

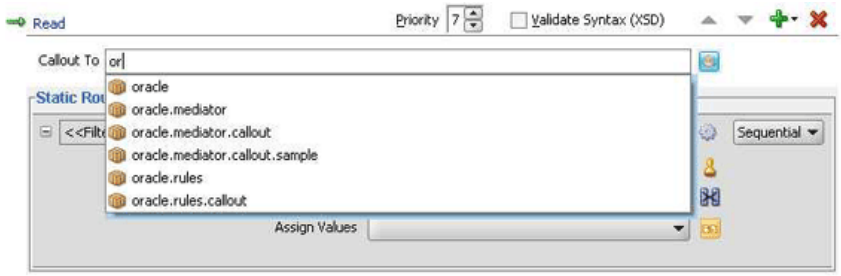
If you want to make the Java callout class available to multiple Mediators, copy the JAR file containing the Java class to the `$DOMAIN_HOME/lib` folder.

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 20-38. The auto-completion feature of Oracle JDeveloper completes the address and the classes in the current project.

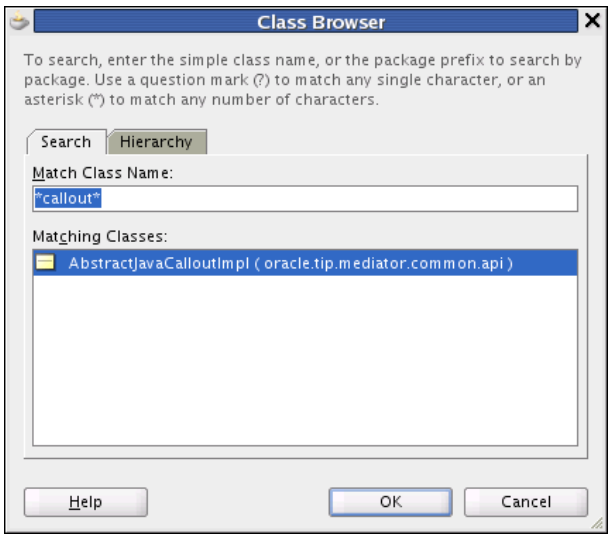
Figure 20-38 Callout To Field



- 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 20-39.

Figure 20-39 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 Mediator and use a filter expression in the same Mediator, you must set the root element for the payload, as shown in the following example:

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

To enable domain value map and cross reference functions:

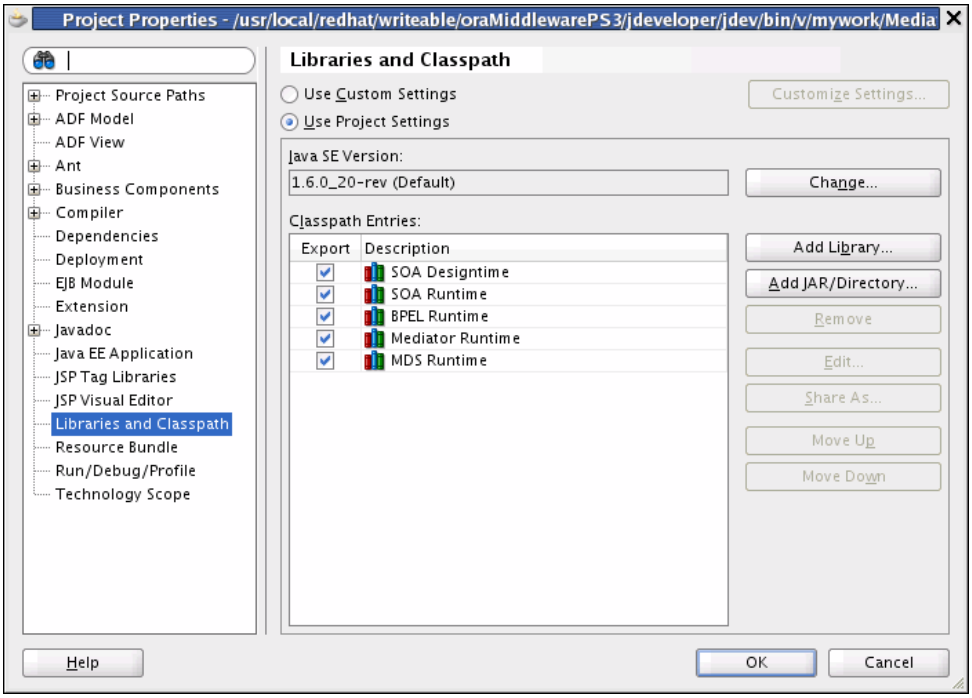
To use domain value map functions or cross reference functions in a Java callout, you must add the `soa-xpath-exts.jar` file to the project and import the necessary Java classes into your code.

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 20-40.

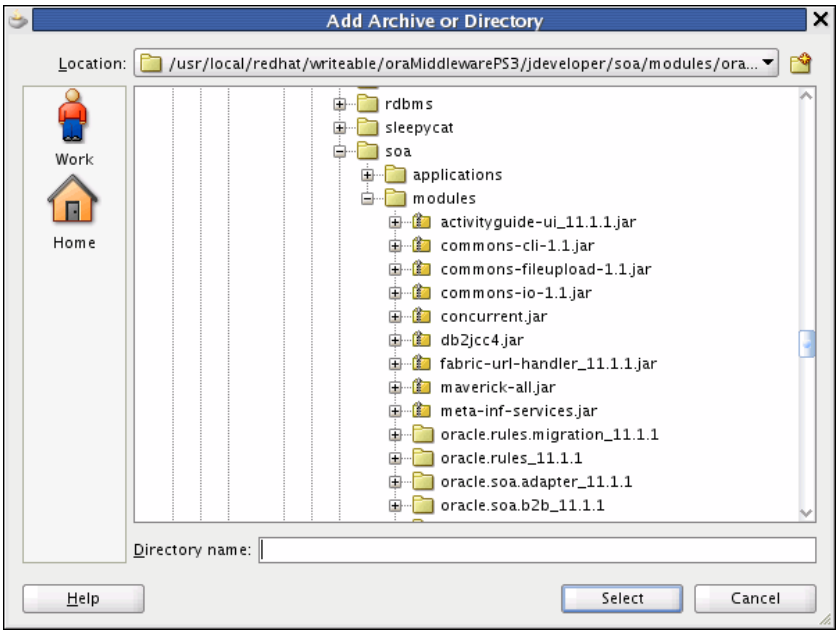
Figure 20-40 Libraries and Classes on the Project Properties Dialog



4. Click **Add JAR/Directory**.

   The **Add Archive or Directory** dialog appears, as shown in Figure 20-41.

Figure 20-41 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**.

Note:

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`

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 20-5 lists and describes the methods in the `oracle.tip.mediator.common.api.IjavaCallout` interface.

Table 20-5 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 Mediator starts executing the cases. You can customize this method to include validations and enhancements.
`preRoutingRule`	This method is called before Mediator starts executing any particular case. You can customize this method to include case-specific validations and enhancements.
`preCallbackRouting`	This method is called before Mediator finishes executing callback handling. You can customize this method to perform callback auditing and custom fault tracking.
`postRouting`	This method is called after Mediator finishes executing the cases. You can customize this method to perform response auditing and custom fault tracking. Post-processing methods are called after all sequential routing rules are executed and do not wait for parallel routing rules to complete.
`postRoutingRule`	This method is called after Mediator starts executing the cases. You can customize this method to perform response auditing and custom fault tracking.

Table 20-5 (Cont.) Description of Methods in the IjavaCallout Interface

Method	Description
`postCallbackRouting`	This method is called after Mediator finishes executing callback handling. You can customize this method to perform callback auditing and custom fault tracking.

Note:

If you change the message properties of a Mediator by using 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 Mediator assignment functionality. For example, if you are changing the `jca.file.FileName` property in a Java callout, then you must update the Mediator assignment statement as follows:

```
<assign>
<copy target="$out.property.jca.file.FileName"
expression="$in.property.jca.file.FileName"/>
</assign>
```

Table 20-6 discusses the methods in the `CalloutMediatorMessage` interface.

Table 20-6 Description of Methods in the CalloutMediatorMessage Interface

Method	Description
`addPayload`	This method sets a payload of the Mediator messages.
`addProperty`	This method adds a property to the Mediator messages.
`addHeader`	This method adds a header to the Mediator messages.
`getProperty`	This method retrieves Mediator message properties by providing the property name.
`getProperties`	This method retrieves Mediator message properties.
`getId`	This method retrieves the instance ID of the Mediator messages. This instance ID is the Mediator instance ID created for that particular message.
`getPayload`	This method retrieves a payload of the Mediator messages.
`getHeaders`	This method retrieves a header of the Mediator messages.
`getComponentDN`	This method retrieves a componentDN for the Mediator service component.

Note:

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

  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.

- Details of the processing occurring within the Java callout are not displayed in the Mediator audit trail screen.

Sample Java Callout Class

The following example shows a sample Java callout class:

```
package qa.as11tests.javacallout;

import com.collaxa.cube.persistence.dto.XmlDocument;

import com.oracle.bpel.client.NormalizedMessage;

import java.util.logging.Logger;
import java.util.Map;
import java.util.Iterator;

import oracle.tip.mediator.common.api.CalloutMediatorMessage;
import oracle.tip.mediator.common.api.ExternalMediatorMessage;
import oracle.tip.mediator.common.api.IJavaCallout;
import oracle.tip.mediator.common.api.MediatorCalloutException;
import oracle.tip.mediator.metadata.CaseType;
import oracle.tip.mediator.utils.XmlUtils;

import oracle.tip.pc.services.functions.ExtFunc;

import oracle.xml.parser.v2.XMLDocument;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;

public class JavaCalloutSanity implements IJavaCallout {
    Logger logger = Logger.getLogger("Callout");
    public JavaCalloutSanity() {    }

    public void initialize(Logger logger) throws MediatorCalloutException {
        this.logger = logger;
        this.logger.info("Initializing...");
    }
    public boolean preRouting(CalloutMediatorMessage calloutMediatorMessage) {
        System.out.println("Pre routing...");
```

```
        String sPayload = "null";
        String sPayload_org = "null";
        for (Iterator msgIt =
calloutMediatorMessage.getPayload().entrySet().iterator();
             msgIt.hasNext(); ) {
            Map.Entry msgEntry = (Map.Entry)msgIt.next();
            Object msgKey = msgEntry.getKey();
            Object msgValue = msgEntry.getValue();
            if (msgKey.equals("request"))
                sPayload =
XmlUtils.convertDomNodeToString((Node)msgValue);
       }
        sPayload_org = sPayload;
        String tobeReplaced = "CHANGE_THIS";
        String replaceWith = "JAVA_CALLOUT_||_PRE_ROUTING";
        int start = sPayload.indexOf(tobeReplaced);
        StringBuffer sb = new StringBuffer();
        sb.append(sPayload.substring(0, start));
        sb.append(replaceWith);
        sb.append(sPayload.substring(start + tobeReplaced.length()));
        String changedPayload = sb.toString();
        String uid;
        try {
            uid = ExtFunc.generateGuid();
        } catch (Exception e) {
        }
              XMLDocument changedoc;
        try {
            changedoc = XmlUtils.getXmlDocument(changedPayload);
            String mykey = "request";
            calloutMediatorMessage.addPayload(mykey,
                                              changedoc.getDocumentElement());
            //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.getDocumentElement());
                // 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.getDocumentElement());
            // 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;
    }
}
```

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. Dynamic routing enables you to dynamically route messages at runtime from a mediator to multiple target services, based on the message content. You can use Domain Value Maps (DVMs) or Decision Components (Business Rules) to override static routes at runtime.

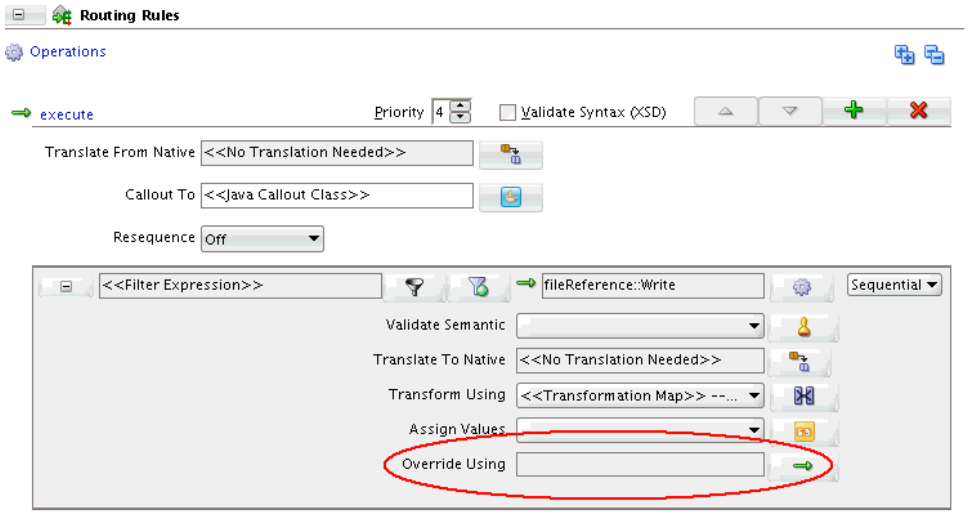
How to Dynamically Override a Static Routing Rule Using a DVM

You can use a Domain Value Map (DVM) to dynamically override an existing static routing rule. You can create a new DVM, or use an existing DVM to override mediator routing options.

To override a static route using DVM:

1. Double-click the mediator component to open the Mediator Editor.
2. Under the Routing Rules section, scroll down to the routing rule that you want to modify.
3. To the right of the **Override Using** field, click the button, identified by the green arrow. Figure 20-42 shows the Override Using field.

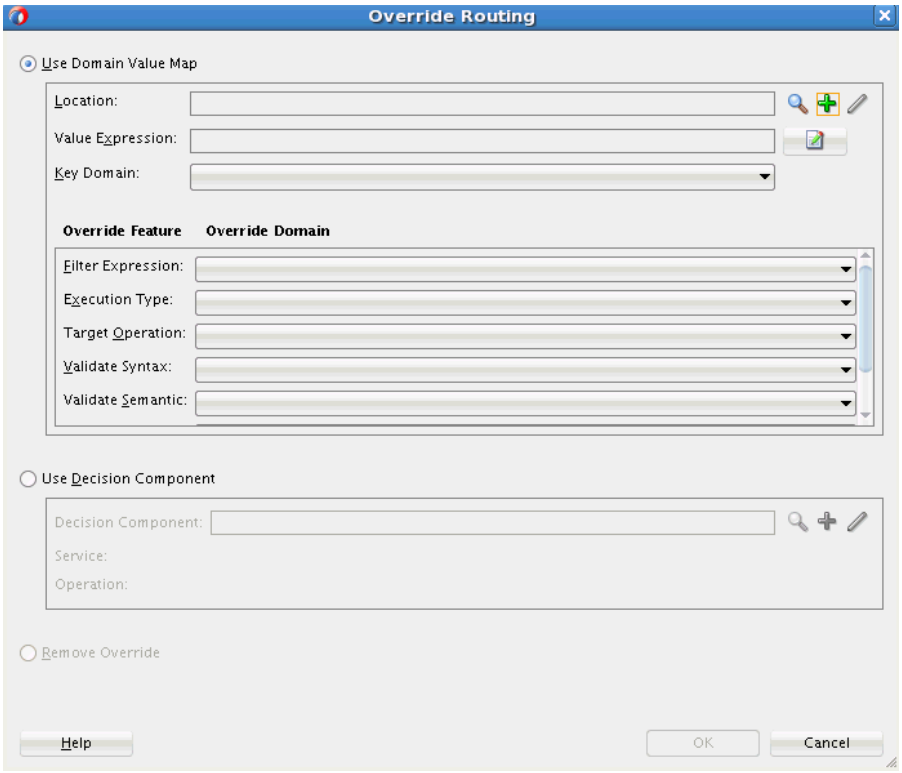
Figure 20-42 Override Using Field Under Routing Rules



The Override Routing dialog appears.

4. Select **Use Domain Value Map** to create or use a domain value map. Figure 20-43 shows the Override Routing dialog.

Figure 20-43 Override Routing Dialog



5. To the right of the Location field, click **Create new DVM file,** identified by the green plus (+) icon, to create a new DVM file. The Create Domain Map Value dialog appears.

Note:

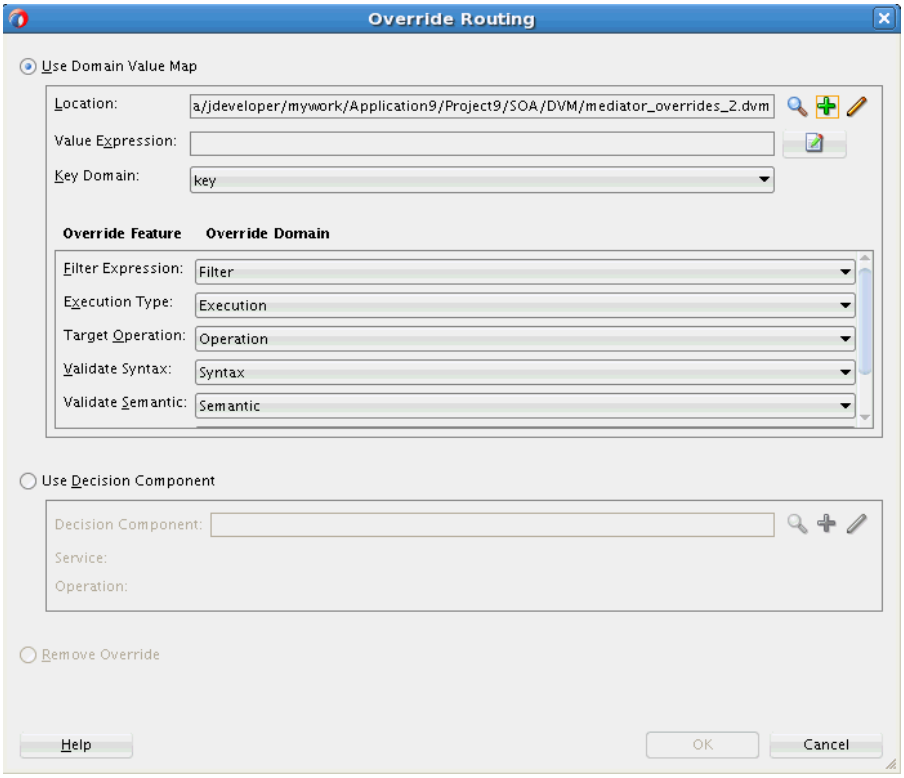
You can also choose an existing DVM file by clicking **Find existing DVM file,** identified by the Search icon.

---

6. In the Create Domain Map Value dialog, specify a **DVM Name** and select a **Directory** to store the DVM file. Click **OK.** The **DVM File Created** dialog appears.

7. Click **OK** to confirm. The Override Routing dialog is now populated with the details of the new DVM. shows the Override Routing dialog after creating a new DVM.

Figure 20-44 New Domain Value Map Details

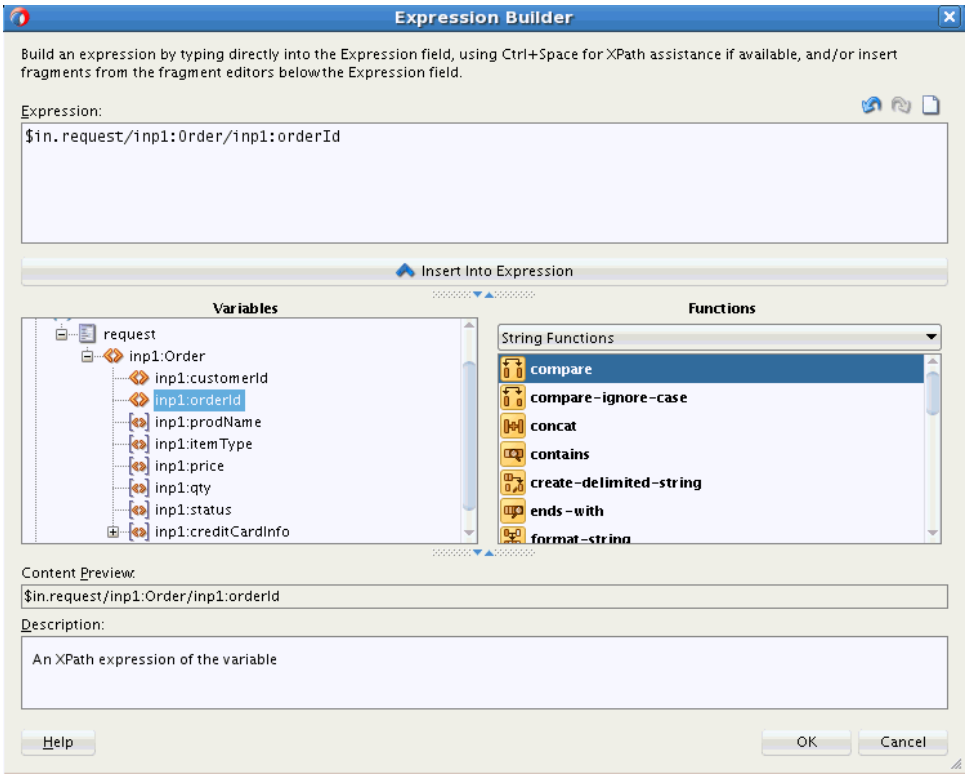


A new domain is created for each feature of the mediator that can be overridden. For example, as shown in Figure 20-44, the Filter domain is created for the mediator Filter Expression.

8. Select a **Key Domain** corresponding to the lookup column for the DVM.

9. To the right of the **Value Expression** field, click the **Invoke Expression Builder** icon to specify a value expression corresponding to the key domain. The Expression Builder dialog appears.

Figure 20-45 Expression Builder



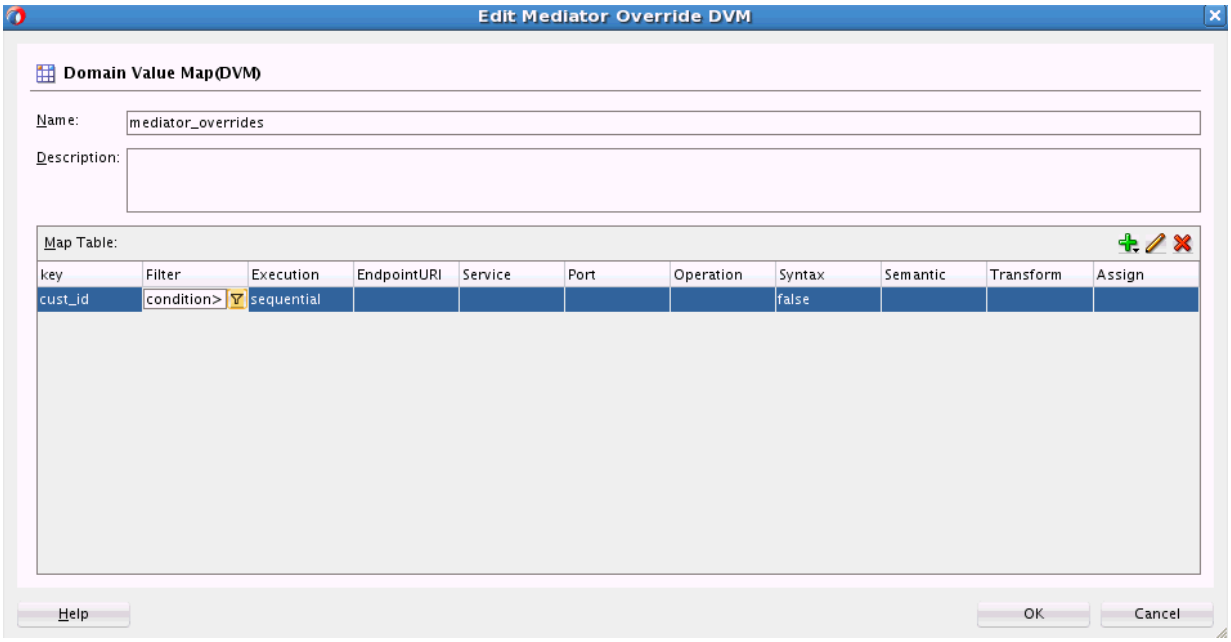
10. Build the expression corresponding to the value expression for the domain key, and click **OK**. You can use the Help button for more information on the Expression Builder.

11. Click **OK** in the Override Routing dialog.

To add a new domain to the DVM:

1. In the Override Routing dialog (Figure 20-44), click the **Open DVM file editor** icon to the right of the Location field. The Edit Mediator Override DVM dialog appears. Figure 20-46 shows the Edit Mediator Override DVM dialog.

Figure 20-46 Edit Mediator Override DVM Dialog



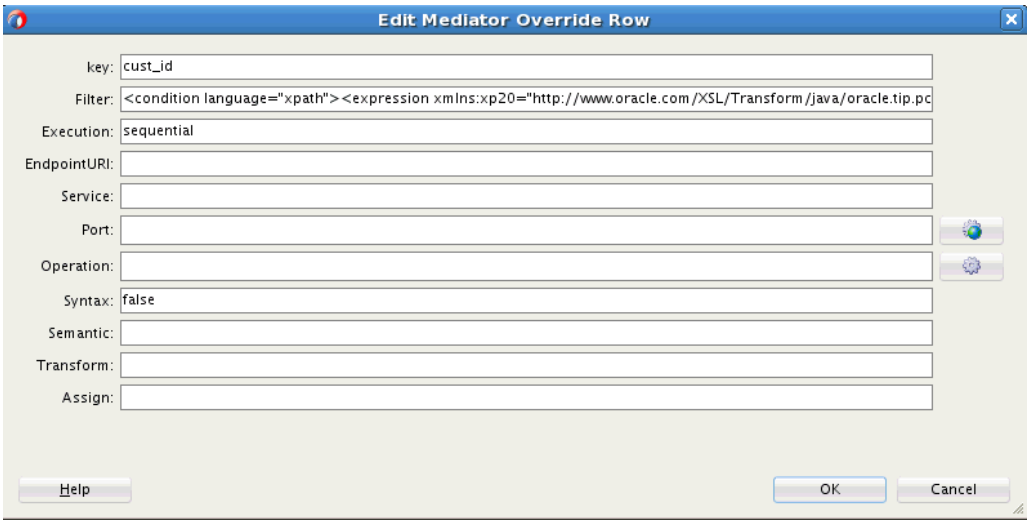
2. If required, edit the **Name** and **Description** of the DVM.
3. Under Map Table, click the **Add Domain/Values** icon identified by the green plus (+) icon. A pop-up menu appears.
4. To add a new domain or column, select **Add Domain**. The Create Domain dialog appears.
5. Specify a **Name** for the new domain. Use the Help button for more details on the Create Domain process.

   Click **OK**

To add a new row to the DVM:

1. In the Override Routing dialog (Figure 20-44), click the **Open DVM file editor** icon to the right of the Location field. The Edit Mediator Override DVM dialog appears (Figure 20-46).
2. Under Map Table, click the **Add Domain/Values** icon identified by the green plus (+) icon. Select **Add Domain Values** from the pop-up menu that appears.
3. You can click each row item to edit it. Alternatively select the row and click the **Edit Domain/Values** icon to edit the row. The Edit Mediator Override Row dialog appears. Figure 20-47 shows the Edit Mediator Override Row dialog.

Figure 20-47 Edit Mediator Override Row Dialog



4. Edit the fields, as desired. The usual mediator tools are available to assist you with the editing. For example, clicking the Transform button next to the Transform domain enables you to create a transformation map. After the edits are complete, click **OK.**

To delete a domain from the DVM:

1. In the Override Routing dialog (Figure 20-44), click the **Open DVM file editor** icon to the right of the Location field. The Edit Mediator Override DVM dialog appears (Figure 20-46).
2. To delete a DVM row, select the row and click the **Remove Domain/Values** icon.

To delete a row from the DVM:

1. In the Override Routing dialog (Figure 20-44), click the **Open DVM file editor** icon to the right of the Location field. The Edit Mediator Override DVM dialog appears (Figure 20-46).
2. To delete a DVM column, select the column and click the **Remove Domain/Values** icon.

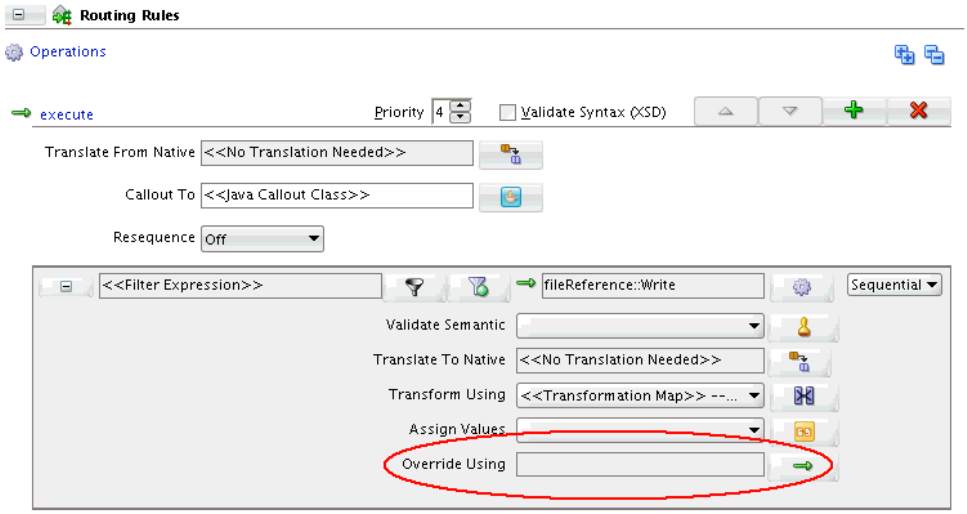
How to Dynamically Override a Static Routing Rule Using a Decision Component

You can use a decision component, or business rule, to dynamically override an existing static routing rule. You can create a new decision component, or use an existing decision component to override mediator routing options.

To override a static route using a Decision Component:

1. Double-click the mediator component to open the Mediator Editor.
2. Under the Routing Rules section, scroll down to the routing rule that you want to modify.
3. To the right of the **Override Using** field, click the button, identified by the green arrow. Figure 20-48 shows the Override Using field.

Figure 20-48 Override Using Field Under Routing Rules



The Override Routing dialog appears.

4. Select **Use Decision Component** to create or use a decision component.

5. To the right of the Decision Component field, click **Create Decision Service,** identified by the green plus (+) icon, to create a new decision service component. The Create Decision Service dialog appears.

Note:

You can also choose an existing decision service component file by clicking **Find existing decision service component,** identified by the Search icon.

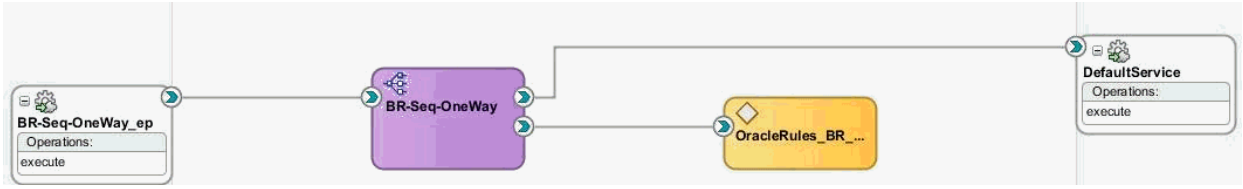
---

6. Specify a **Component Name** for the decision component and a **Service Name** for the service. Click **OK.** The new decision service component is created, and you are returned to the Override Routing dialog. The dialog now contains the details for the decision service component.

   This creates a new business rule service component that is wired to the Mediator service component within the SOA composite of the Mediator service component.

   If you look at the design view of the composite, you can see a business rule component wired to the mediator in addition to the static reference wiring. Figure 20-49 shows the design view for a sample composite.

Figure 20-49 Mediator Connected to a Business Rule Component



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 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.

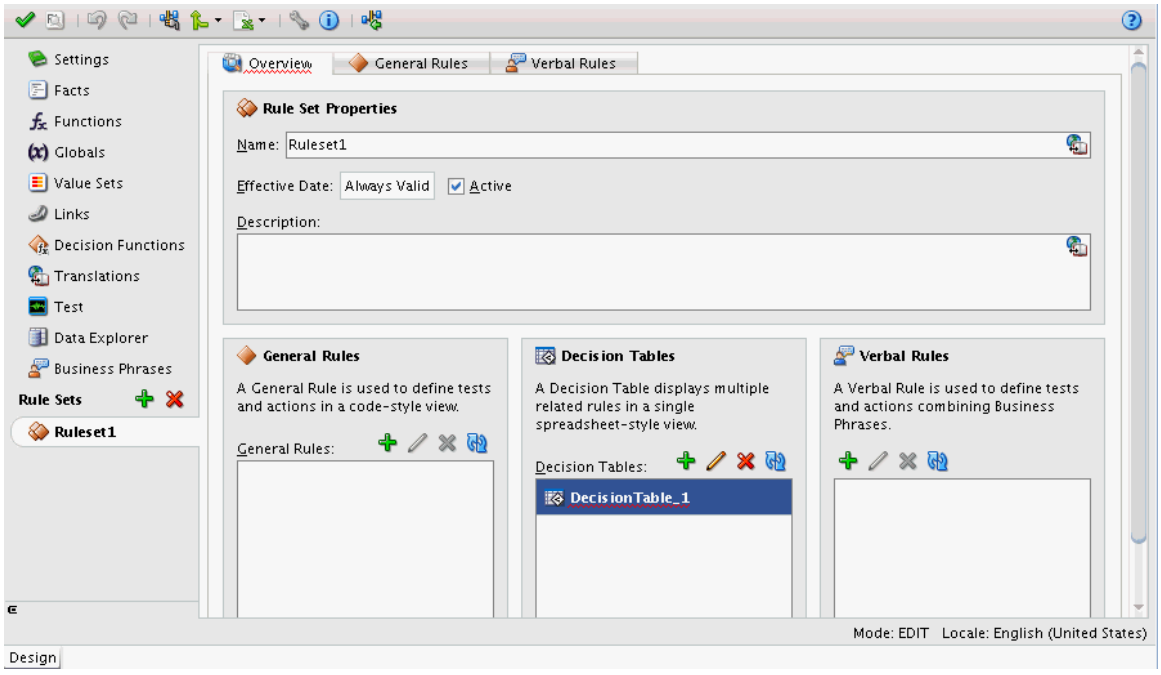
- Decision Service

  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 Mediator service component to evaluate the decision table.

To edit a decision component:

1. In the Override Routing dialog (Figure 20-44), click the **Open Decision Component Editor** icon to the right of the Decision Component field. The Decision Component Editor appears, as shown in Figure 20-50.

Figure 20-50 Decision Component Editor



2. Under Decision Tables, select the decision table and click **Edit** to edit the decision table.

   See Getting Started with Oracle Business Rules for more information on working with decision tables and business rules.

How to Remove an Existing Dynamic Routing Rule

You can remove a DVM or Decision Component based routing rule override.

To remove a dynamic routing rule override:

1. Double-click the mediator component to open the Mediator Editor.
2. Under the Routing Rules section, scroll down to the routing rule that you want to modify.
3. To the right of the **Override Using** field, click the button, identified by the green arrow. Figure 20-48 shows the Override Using field.

   The Override Routing dialog appears.
4. Select **Remove Override** to remove any static routing rule overrides.
5. Click **OK.**

What You May Need to Know About Using Dynamic Routing Rules

Note the following limitations on using dynamic routing rules with Mediator:

- All possible message patterns are supported (Synchronous, Asynchronous, Synchronous-Asynchronous, and One-Way).
- Event publishers and echo cannot have dynamic routing rules associated with them.
- Static rule overrides are applicable only for requests, and not for responses. If you must override a response, you must route it to another mediator and override it as a request.
- When overriding a target port, the overriding port must be of the same port type.

How to Define Default Routing Rules

Mediator processes messages depending on the conditions specified in the routing rules. In some cases, a 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 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.

Note:

- Default rules are available only for static routing rules.
- You cannot specify a default routing rule for a Mediator service component with dynamic routing rules because you cannot define both static and dynamic routing rules in the same Mediator service component.

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 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. The following example provides details:

```
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 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 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 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 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 Mediator service component. Similarly, if a 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.

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.

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.

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 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.

Default Rule: Mediator .mplan File

To set a routing rule as the default one, click the **Set as Default Routing Rule** icon shown on Figure 20-2. The `.mplan` file changes, as shown in Figure 20-51.

Figure 20-51 .mplan File of a Mediator with a Default Routing Rule

```
<Mediator name="Mediator1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" wsdlTargetNamespace=
   <operation name="execute" deliveryPolicy="AllOrNothing" priority="4" validateSchema="false">
      <switch>
         <case executionType="sequential" name="SOAPReference_3.sayHello" defaultRule="true">
            <action>
               <invoke reference="SOAPReference_3" operation="sayHello">
                  <onReply/>
               </invoke>
            </action>
         </case>
         <case executionType="sequential" name="SOAPReference_2.execute">
            <action>
               <dvm location="../DVM/mediator_overrides.dvm" sourceColumnName="key"
                    sourceValueExpression="$in.part1/inp1:PurchaseOrder/ID">
                  <dynamicParam name="port" value="Port"/>
                  <dynamicParam name="operation" value="Operation"/>
```

Mediator Routing Use Cases

Two tutorials are available that give you step-by-step instructions for creating two of the Mediator sample projects provided on the `Oracle SOA Suite samples page`. They illustrate how to define routing rules for the Mediators you create. You can download the tutorials from `http://java.net/projects/oraclesoasuite11g/downloads/download/Mediator/Tutorials/med_rr_tutorial.pdf`.

21

Working with Multiple Part Messages in Oracle Mediator

This chapter describes how to define routing rules for multiple part (multipart) messages for an service component, including defining filters, transformations, and validations.

This chapter includes the following sections:

- Introduction to Mediator Multipart Message Support
- Working with Multipart Request Messages

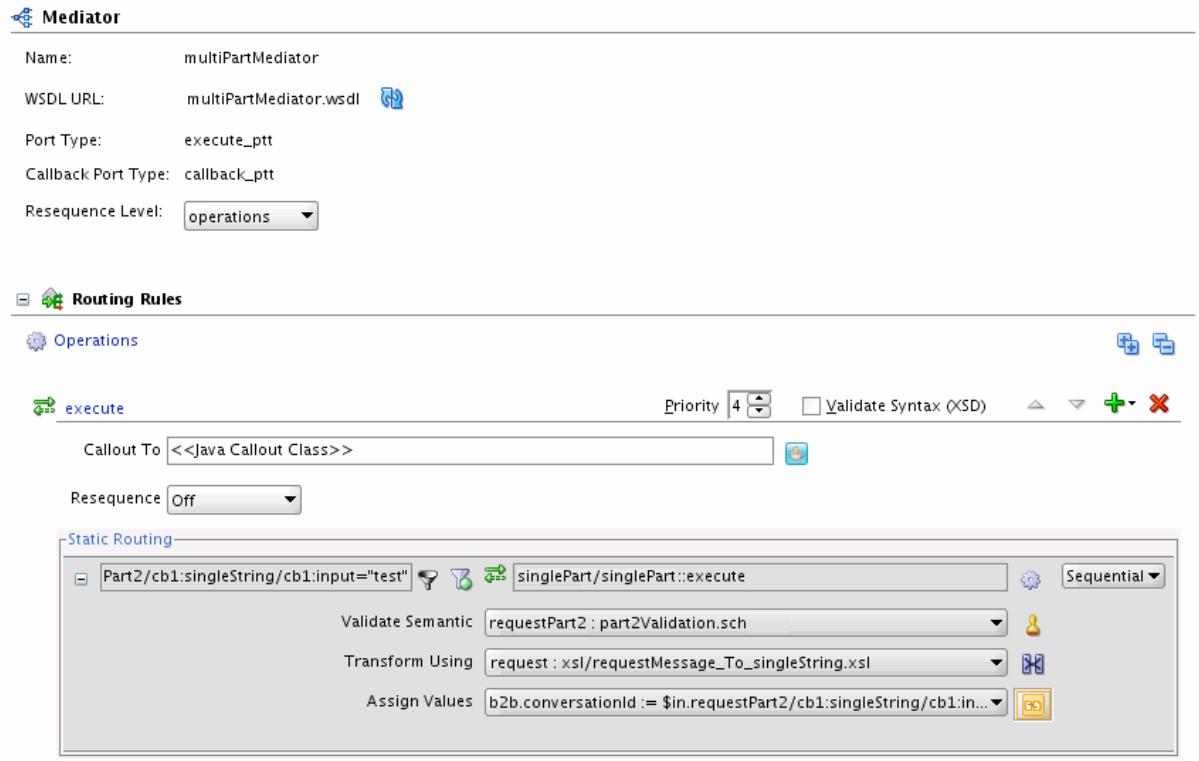
For more information on routing rules, see Creating Routing Rules .

Introduction to Mediator Multipart Message Support

Mediator includes support for 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 looks similar to Figure 21-1.

Figure 21-1 Mediator Editor for a Multipart Source



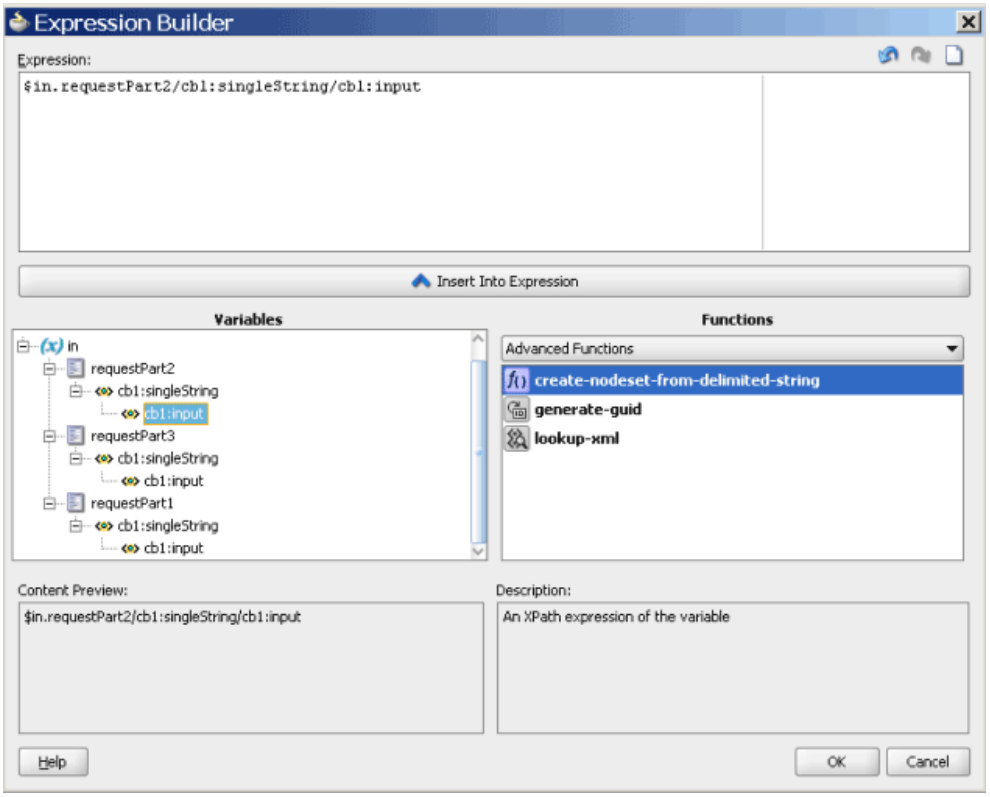
Working with Multipart Request Messages

This section describes how to work with different types of multipart messages.

How to Specify Filter Expressions for Multipart Request Messages

If you specify a filter expression for a multipart message, then the Expression Builder displays all message parts under the **in** variable, as shown in Figure 21-2:

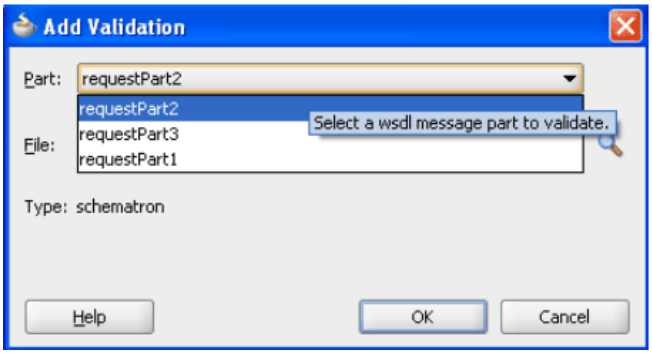
Figure 21-2 Expression Builder for a Multipart Request Source



How to Add Validations for Multipart Request Messages

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 21-3. You specify a Schematron file for each request message part. Oracle Mediator then processes the Schematron files for the parts.

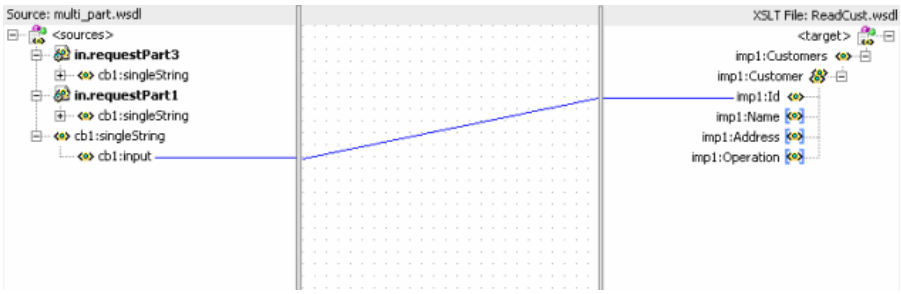
Figure 21-3 Add Validation Dialog for a Multipart Request Source



How to Create Transformations for Multipart Request Messages

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 21-4:

Figure 21-4 XSLT Mapper for a Multipart Request Source



How to Assign Values for Multipart Request Messages

If you assign values using a source expression and invoke the Expression Builder from the Assign Value dialog, the Expression Builder displays all message parts under the **in** variable, as shown in Figure 21-2. This is the same as specifying filter expressions.

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.

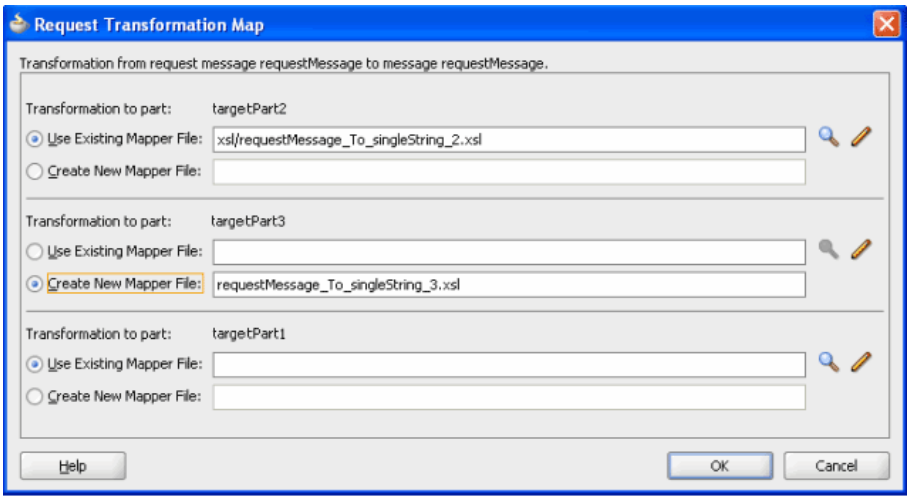
Note:

You cannot specify filter expressions or add validations for reply, fault, and callback messages.

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 Mediator creates and coordinates a separate mapper file for each target part, as shown in Figure 21-5:

Figure 21-5 Request Transformation Map for a Multipart Routing Target



22

Using Oracle Mediator Error Handling

This chapter describes the error handling capabilities of Oracle Mediator and provides instructions for defining error handling for both business faults and system faults.

This chapter includes the following sections:

Introduction to Mediator Error Handling

Mediator provides sophisticated error handling capabilities that enable you to configure a Mediator service component for error occurrences and corresponding corrective actions. Error handling enables a Mediator to handle errors that occur during the processing of messages and also the exceptions returned by outside web services. You can handle both business faults and system faults with Mediator.

Business faults are application-specific and are explicitly defined in the service WSDL file. You can handle business faults by defining the fault handlers in Oracle JDeveloper at design time. System faults occur because of some problem in the underlying system such as a network not being available. Mediator provides fault policy-based error handling for system faults.

Fault policies enable you to handle errors automatically or through human intervention. Mediator fault policy-based error handling consists of the following three components:

- Fault policies
- Fault bindings
- Error groups

Fault Policies

A fault policy defines error conditions and corresponding actions. Fault policies are defined in the `fault-policies.xml` file, which should be created based on the XML schema shown in Schema Definition File for fault-policies.xml .

Fault policies for sequential routing rules are handled differently than for parallel routing rules, as described below:

- Due to the single threading of sequential routing rules, only three actions (Abort, Rethrow, and Java) are supported for handling errors, and the specified actions are executed immediately in the caller's thread.
- Mediator messages are not persisted in sequential routing.
- Asynchronous and one-way Mediator components cannot handle system faults thrown from other SOA Suite components, such as a BPEL business process.

For more information about available error handling actions, see Actions.

Note:

Fault policies are not supported for the following:

- Callback execution failures
- Fault Handler action failures
- Resequencer failures

A sample fault policy file is shown below:

```
<?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.

Conditions

Conditions allow you to identify error or fault conditions and then specify the actions 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 the following example:

```
<Conditions>
      <faultName xmlns:medns="http://schemas.oracle.com/mediator/faults"
       name="medns:mediatorFault">
          <condition>
            <test>contains($fault.mediatorErrorCode,"TYPE_DATA_TRANSFORMATION")</
test>
             <action ref="ora-java"/>
          </condition>
      </faultName>
      <faultName xmlns:medns="http://schemas.oracle.com/mediator/faults"
name="medns:mediatorFault">
        <condition>
         <test>contains($fault.mediatorErrorCode, "TYPE_FATAL_MESH")</test>
          <action ref="ora-retry"/>
        </condition>
      </faultName>
      <faultName xmlns:medns="http://schemas.oracle.com/mediator/faults"
name="medns:mediatorFault">
        <condition>
         <test>contains($fault.mediatorErrorCode,"TYPE_DATA_ASSIGN")</test>
          <action ref="ora-retry-crm-endpoint"/>
        </condition>
      </faultName>
</Conditions>
```

Identifying Fault Types Using Conditions

You can categorize the faults that can be captured using conditions into the following types:

- Mediator-specific faults

  For all Mediator-specific faults, the Mediator service engine throws only one fault, namely `{http://schemas.oracle.com/mediator/faults}mediatorFault`. Every Mediator fault is wrapped into this fault. The errors or faults generated by a Mediator can be captured by using the format shown in the following example:

  ```
  <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. The following example provides details:

```
<faultName xmlns:ns1="http://xmlns.oracle.com/Customer"
 name="ns1:InvalidCustomer"> <!-- Qname of Business/SOAP fault -->
       <condition>
         <test>
contains($fault.<PART_NAME>/custid, 1011)
 </test>
<!-- xpath condition based on fault payload -->
         <action ref="ora-retry"/>
       </condition>
   </faultName>
```

When a reference service returns a business fault, the fault can be handled in the Mediator 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 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 the following example:

```
<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>
```

Actions

Actions specify the tasks to perform when an error occurs. Mediator supports retry, human intervention, abort, and Java code actions for parallel routing rules. For sequential routing rules, fault policies can contain these actions: abort, rethrow, and Java code.

If retry or human intervention action is chosen with sequential routing rules, the fault goes back to the caller directly, and the policy is not applied. The fact that an incompatible action was chosen is recorded in the log. This is consistent with BPEL fault policy behavior. It is the responsibility of the caller to handle the fault. If the caller is an adapter, 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 *Understanding Technology Adapters*.

Fault policy actions are described in the following sections.

Retry Action

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 22-1 describes the options available with the retry action. Retry actions are applicable to parallel routing rules only.

Table 22-1 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.

The following example shows how to specify the retry action:

```
<Action id="ora-retry">
<retry>
<retryCount>3</retryCount>
<retryInterval>2</retryInterval>
<exponentialBackoff/>
<retryFailureAction ref="ora-java"/>
<retrySuccessAction ref="ora-java"/>
</retry>
</Action>
```

If you set the retry interval in the fault policy to a duration 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>
```

Rethrow Action

Rethrow executes the fault policy in the caller's thread and returns the original exception to the user.

An example of a rethrow action is shown below:

```
<Action id="ora-rethrow-fault"><rethrowFault/></Action>
```

Human Intervention Action

The human intervention action allows you to manually recover the fault by correcting the error (for example, altering the payload) and then manually retrying the message. This action is applicable to parallel routing rules only.

An example of a human intervention action is shown below:

```
<Action id="ora-human-intervention"><humanIntervention/></Action>
```

Abort Action

The abort action enables you to terminate the message flow. This action is applicable to both parallel and sequential routing rules.

When the abort action is executed for a sequential routing rule, the exception `FabricInvocationException` is thrown back to the caller, and the mediator component state changes to terminated. The fault policy is executed in the caller's thread.

An example of an abort action is shown below:

```
<Action id="ora-terminate"><abort/></Action>
```

Java Code Action

The Java code action lets you call a customized Java class that implements the `oracle.integration.platform.faultpolicy.IFaultRecoveryJavaClass` interface. This action is applicable to both parallel and sequential routing rules. The following example shows how Java code actions can be implemented.

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.

The Java code action first looks for the implemented class in the domain class library. If the class is not found there, the action looks in the Composite Application's class library.

```
 <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 a sequential routing rule fault policy, the `returnValue` action must be one of Abort, Rethrow, or Java action. If the `returnValue` is other than these valid values, then the `defaultAction` is checked. If the `defaultAction` is also not a valid action (Abort, Rethrow, or Java action), then no action is performed by default, and the original fault is thrown back to the caller.

```
oracle.integration.platform.faultpolicy.IFaultRecoveryJavaClass {

    public void handleRetrySuccess(IFaultRecoveryContext ctx);
    public String handleFault(IFaultRecoveryContext ctx);
}
```

```
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();
}
```

Mediator Service Engine Implementation

The following example shows the Oracle Mediator service engine implementation of the `IFaultRecoveryContext` interface.

```
package oracle.tip.mediator.common.error.recovery;
public class MediatorRecoveryContext implements IFaultRecoveryContext{
   ...
}
```

You can use the methods shown in the following example to retrieve additional Mediator-specific data available with the `MediatorRecoveryContext` class:

```
public CommonFault getACommonFault()
public CalloutMediatorMessage getMediatorMessage()
```

The following example shows how to retrieve data 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;
```

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 Schema Definition File for fault-bindings.xml .

Fault policies can be created at the following levels:

- **Composite**: You can define one fault policy for all Mediator components in a composite. You can specify this level in the following way:

```
<composite faultPolicy="ConnectionFaults"/>
```

- **Component**: You can define a fault policy exclusively for a Mediator service component. A component-level fault policy overrides the composite-level fault policy. You can specify this level as shown in the following example:

```
<component faultPolicy="ConnectionFaults">
        <name>Component1</name>
        <name>Component2</name>
</component>
```

- Reference: You can define a fault policy for the references of a Mediator component. You can specify this level as shown in the following example:

```
<reference faultPolicy="policy1">
    <name>DBAdapter3</name>
  </reference>
```

Note:

The level of precedence for fault policies is Reference -> Component -> Composite.

Note:

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

A sample fault binding file is shown in the following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<faultPolicyBindings version="2.0.1"
 xmlns="http://schemas.oracle.com/bpel/faultpolicy"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <composite faultPolicy="ConnectionFaults"/>
</faultPolicyBindings>
```

Error Groups in Mediator

You can specify an action for an error type or error group while defining the conditions in a fault policy. In the previous examples, `medns:mediatorFault` indicates that the error is a Mediator error, whereas `medns:TYPE_FATAL_MESH` refers to an error group. An error group consists of one or more child error types. `TYPE_ALL` is an error group that contains all Mediator errors.

The following list describes various error groups contained in the `TYPE_ALL` error group:

- `TYPE_DATA`: Contains errors related to data handling.
  - `TYPE_DATA_ASSIGN`: Contains errors related to data assignment.
  - `TYPE_DATA_FILTERING`: Contains errors related to data filtering.

  - TYPE_DATA_TRANSFORMATION: Contains errors that occur during a transformation.
  - TYPE_DATA_VALIDATION: Contains errors that occur during payload validation.
- TYPE_METADATA: Contains errors related to Mediator metadata.
  - TYPE_METADATA_FILTERING: Contains errors that occur while processing the filtering conditions.
  - TYPE_METADATA_TRANSFORMATION: Contains errors that occur while getting the metadata for a transformation.
  - TYPE_METADATA_VALIDATION: Contains errors that occur during validation of metadata for Mediator (`.mplan` file).
  - TYPE_METADATA_COMMON: Contains other errors that occur during the handling of metadata.
- TYPE_FATAL: Contains fatal errors that are not easily recoverable.
  - TYPE_FATAL_DB: Contains database-related fatal errors, such as a `Datasource not found` error.
  - TYPE_FATAL_CACHE: Contains Mediator cache-related fatal errors.
  - TYPE_FATAL_ERRORHANDLING: Contains fatal errors that occur during error handling such as `Resubmission queues not available`.
  - TYPE_FATAL_MESH: Contains fatal errors from the Service Infrastructure such as `Invoke service not available`.
  - TYPE_FATAL_MESSAGING: Contains fatal messaging errors arising from the Service Infrastructure.
  - TYPE_FATAL_TRANSACTION: Contains fatal errors related to transactions such as `Commit can't be called on a transaction which is marked for rollback`.
  - TYPE_FATAL_TRANSFORMATION: Contains fatal transformation errors such as 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.

Using Error Handling with Mediator

You can enable error handling for an Oracle Mediator by using the `fault-policies.xml` and `fault-bindings.xml` files.

How to Use Error Handling for a Mediator Service Component

To use error handling for a Mediator service component:

1. Create a `fault-policies.xml` file based on the schema defined in Schema Definition File for fault-policies.xml .
2. Create a `fault-bindings.xml` file based on the schema defined in 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.

What Happens at Runtime

All the fault policies for a composite are loaded when the first error occurs. When an error occurs, the Mediator Service Engine checks for the existence of the fault policy files (`fault-policies.xml` and `fault-bindings.xml`). The fault policy bindings are checked to determine the fault policy associated with the component or composite. If a fault policy is associated with the component or composite, then Mediator performs the action defined in the fault policy corresponding to the fault condition. If no fault policy bindings are found for the component or composite, then no action is performed and the behavior is the same as if the fault policies did not exist.

If there is no fault policy defined and the routing rule is executed in parallel, the default action of human intervention is performed. If there is no fault policy defined and the routing rule is executed sequentially, the error is thrown back to the caller.

Note:

All sequential routing transactions that encounter an error are rolled back, even if a fault policy has been used to handle the errors.

For more information about how fault policies are processed, see Actions.

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. Use any of the following ways to recover faults:

- 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 *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

Error Handling XML Schema Definition Files

This section describes the schema files for the `fault-policies.xml` and `fault-bindings.xml` files.

Schema Definition File for fault-policies.xml

The `fault-policies.xml` file should be based on the XSD file as shown in the following example:

```
<?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>
```

Schema Definition File for fault-bindings.xml

The `fault-bindings.xml` file should be based on the XSD file as shown in the following example:

```
<?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>
```

23

Resequencing in Oracle Mediator

This chapter describes message resequencing concepts in Oracle Mediator, and provides instructions for configuring standard resequencing, first-in/first-out resequencing, and best effort resequencing.

This chapter includes the following sections:

Introduction to the Resequencer

The resequencer in 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 messages to the target services in an orderly manner. The sequencing is performed based on the sequencing strategy selected.

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 a 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 23-1 shows how the Mediator sorts the messages into groups. The order of the messages in each group depends on the type of resequencer used.

Table 23-1 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.

Identification of Groups and Sequence IDs

Groups and sequence IDs are identified through XPath expressions in the message payload and header. 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 23-1, `CustomerId` is the field on which to base instance sequencing and `Type` is the field on which to base grouping.

Figure 23-1 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 not supported for synchronous operations.

Resequencing Order

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

Standard Resequencer

The standard resequencer supports a standard resequencer pattern. The following sections describe the standard resequencer and how it processes messages.

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 Mediator service component are guaranteed to arrive in sequence.

Information Required for Standard Resequencing

When using the standard resequencer in Mediator, you must always specify a group XPath expression and a sequence ID XPath expression. These specify where the 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 timeout period, in seconds, to wait for the expected messages.

The Mediator standard resequencer holds back messages in the Mediator resequencer database until it can produce the right sequence for different groups. This situation means that if for a given group, the message with a particular sequence ID does not arrive within the timeout period, the subsequent 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.

Example of the Standard Resequencer

Table 23-2 shows how groups are formed differently for two different values of the incremental delta.

Table 23-2 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,...

Note:

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.

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.

Overview of the FIFO Resequencer

The FIFO resequencer is useful for applications that need sequencing based on the time the messages arrive to the 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 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.

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 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.

Example of the FIFO Resequencer

Table 23-3 illustrates the behavior of the FIFO resequencer where `msgX(Y,Z)` indicates that the message arrives as message number X to the Mediator service component and the message contains `sequenceID` Y and group Z.

Table 23-3 FIFO Resequencer Behavior

Incoming Messages	Sequenced Messages
msg12(4,c)	msg12(4,c),msg10(3,c),msg06(1,c),msg03(2,c)
msg05(9,a)	msg05(9,a), msg02(7,a), msg10(3,a), msg07(5,a)
msg02(7,a)	
msg10(3,c)	
msg10(3,a)	
msg07(5,a)	
msg06(1,c)	
msg03(2,c)	

As shown in Table 23-3, 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 a 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 Mediator service component instead when it arrives to the adapter service.

Best Effort Resequencer

The Mediator resequencer supports a best effort pattern. The following sections describe the best effort resequencer and how it processes messages.

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 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 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 situation 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.

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 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.

Best Effort Resequencer Message Delivery

It is important to note that the messages to outbound services of the 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.

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 data 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.

Example of Best Effort Resequencing Based on Maximum Rows

Table 23-4 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 Mediator service component and the message contains `sequenceID` Y and `group` Z.

Table 23-4 Best Effort Resequencer Behavior Based on Maximum Rows

Group C	Sequenced Messages
msg03(1,c) msg06(2,c) msg10(3,c) msg12(4,c)	msg12(4,c),msg10(3,c),msg06(2,c),msg03(1,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).

Example of Best Effort Resequencing Based on a Time Window

Table 23-5 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 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 23-5 Best Effort Resequencer Behavior Based on a Time Window

Group C Message/Time	Sequenced Messages
msg01(04)/2:00:00 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	msg03(01), msg06(02), msg04(03), msg01(04), msg02(05), msg09(06), msg05(07), msg08(08), msg12(09), msg11(10), msg10(12), msg07(13)

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.

Configuring the Resequencer

You can configure the resequencer using Oracle JDeveloper. This section describes how to configure the resequencer in Oracle JDeveloper.

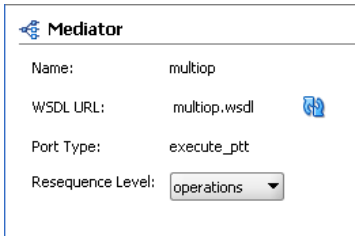
How to Specify the Resequencing Level

You can define resequencing at either the service component level or the operation level. For Mediator service components with only one operation, configuring resequencing at the operation or service component level results in the same behavior. For 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 23-2.

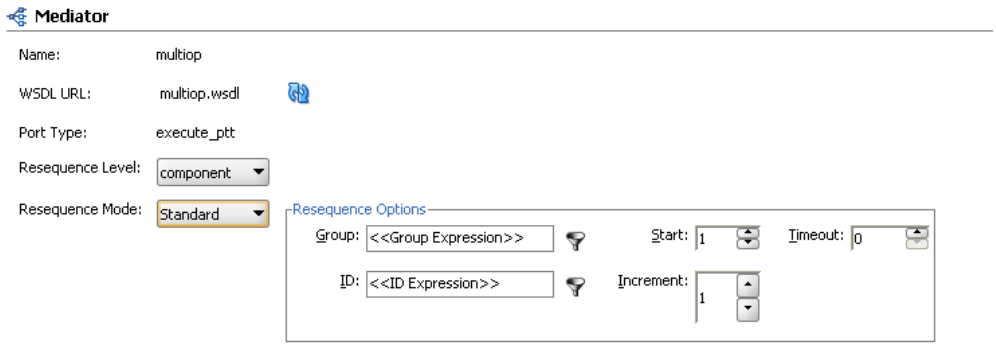
Figure 23-2 Mediator Editor with Resequence Level Field



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 23-3. If the **Resequence Mode** field for an operation is set to **off**, the **Resequence Options** box disappears.

Figure 23-3 Mediator Editor with Resequence Options Section



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

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 How to Specify the Resequencing Level.
2. On the Mediator Editor under either the 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 23-4.

   Figure 23-4 Oracle Mediator with Resequence Mode set to Standard

3. Fill in the fields listed inTable 23-6.

> **Note:**
>
> To specify values for the Group and ID fields, click the **Invoke Expression Builder** button to the right of each field. This launches the Expression Builder, which provides graphical assistance in creating field expressions and adding functions.

Table 23-6 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.	*component_name-operation*	N
ID	The XPath that points to the field in the incoming message on which resequencing is done.	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.	0[1]	N
Start	The starting number of the ID sequence.	1	N
Increment	The increment of the ID sequence.	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 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 Figure 23-5.

Figure 23-5 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.

To configure a best effort resequencer:

1. Set the resequence level as described in How to Specify the Resequencing Level.
2. On the Mediator Editor under either the 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 23-6.

Figure 23-6 Oracle Mediator with Resequence Mode set to Best Effort

3. Fill in the fields listed in Table 23-7 to configure the best effort resequencer.

Note:

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. You must set one control to zero for the other control to be enabled.

---

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 *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

Table 23-7 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.	*component_name-operation*	N
ID	The XPath that points to the field in the incoming message that contains the ID on which resequencing is performed.	N/A	Y

Table 23-7 (Cont.) Best Effort Resequencing Options

Field Name	Description	Default Value	Mandatory
Datatype	The data type of the sequence ID. The ordering process is based on the data type. Supported values are datetime and numeric.	Numeric	Y
Max Rows	Number of in-sequence messages that the resequencer should pick from the data store at a time. You must specify a time window or the maximum rows, but not both. You must set one control to zero for the other control to be enabled.	5	N
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, but not both. You must set one control to zero for the other control to be enabled.	0	N

24

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:

Note:

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 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 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.
- When there are multiple routing rules in a request-response pattern with multiple rules sending a response back to the initial caller, the first response that is received is the one delivered to the caller. The other responses are ignored. Thus, the routing rules that send the response should precede other routing rules that forward the response (if any).

One-way Message Exchange Patterns

In a one-way interaction, the Mediator is invoked, but it does not send a response back to the caller. Depending on the type of routing rule target, the responses, faults, and callbacks are handled as shown in Table 24-1:

Note:

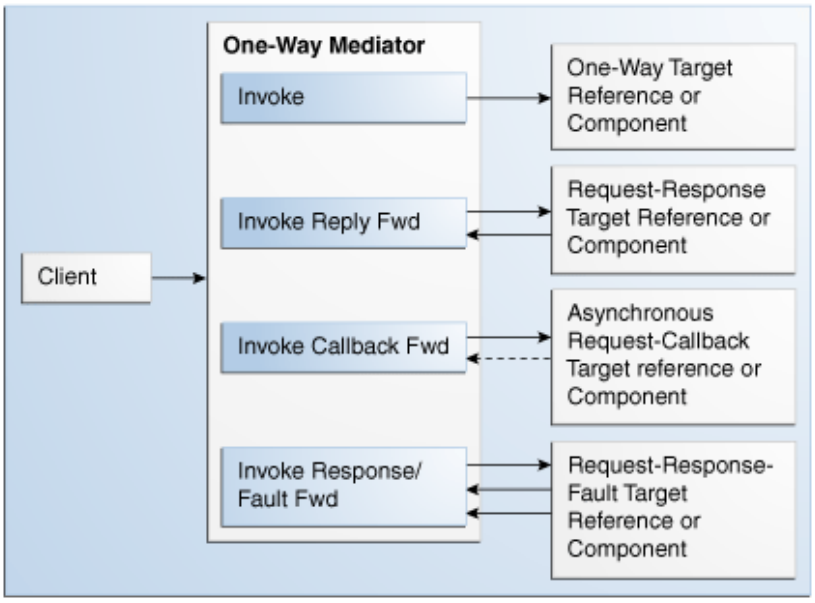
Event subscriptions follow the same exchange pattern as one-way interactions.

Table 24-1 Response When 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 24-1 illustrates the one-way message exchange pattern.

Figure 24-1 One-way Message Exchange Pattern



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 24-2 provides more details on this property.

Table 24-2 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/LocalSan`
`dbox/Project1/ReadFile%2F#wsdl.endpoint(Mediator1/Read_`
`pt)">`
`<property name="one.way.returns.fault"`
`type="xs:string" many="false"`
`override="may">false</property>`
`</binding.ws>`
`</service>`
`. . .`
	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**.

Request-Reply Message Exchange Patterns

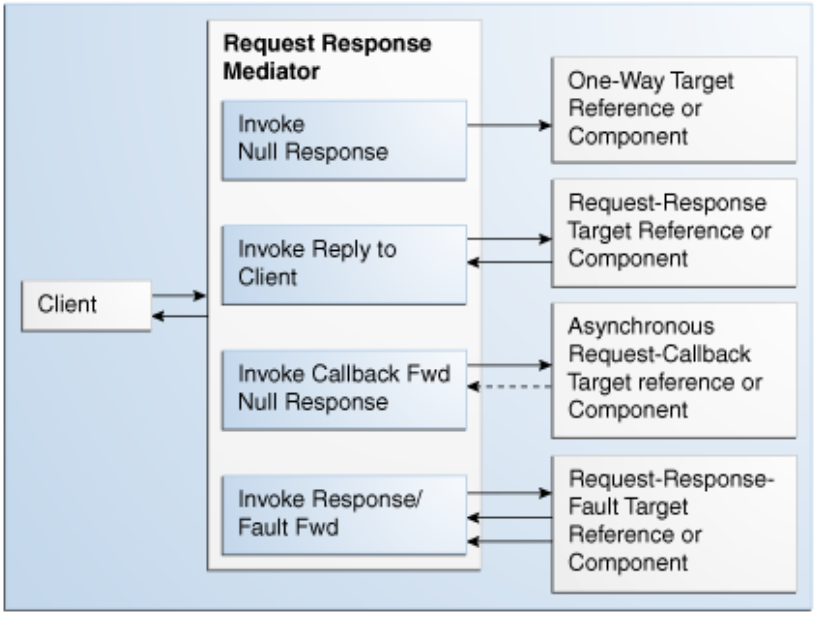
In a request-reply interaction, the Mediator is invoked and sends a reply to the caller. Depending on the type of routing rule target, the responses, faults, and callbacks are handled as shown in Table 24-3:

Table 24-3 Response When 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 24-2 illustrates the request-reply message exchange pattern.

Figure 24-2 Request-Reply Message Exchange Pattern



Request-Reply-Fault Message Exchange Patterns

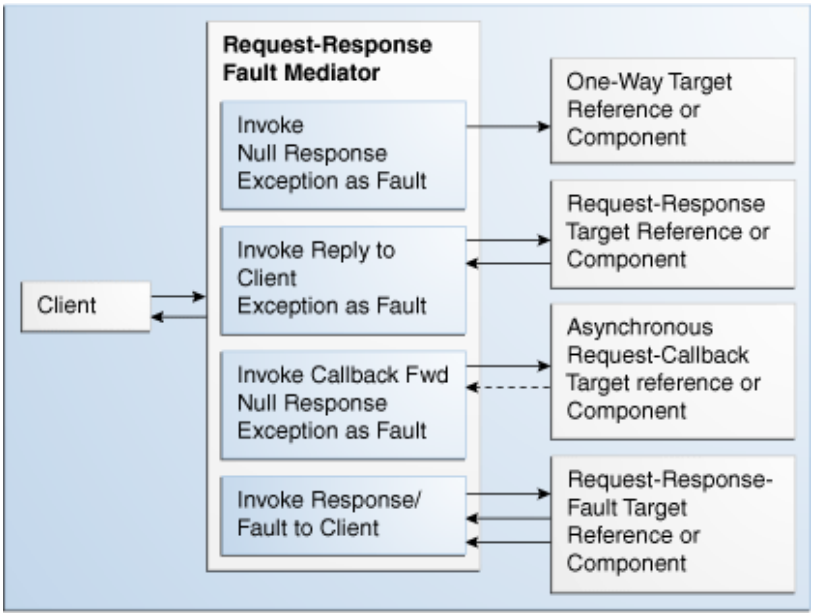
In a request-reply-fault interaction, the Mediator is invoked and sends a reply and one or more faults back to the caller. Depending on the type of routing rule target, the responses, faults, and callbacks are handled as shown in Table 24-4:

Table 24-4 Response When 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. Mediator returns `null` when there is no response to be sent.
Request Response	The response is sent back to the caller. Any exception in Mediator message processing may result in a fault.
Request Response Fault	The response and fault are sent back to the caller. Any exception in 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. 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 Mediator message processing may result in a fault.
Request Response Fault Callback	The response and fault are sent back to the caller. Any exception in Mediator message processing may result in a fault.

Figure 24-3 illustrates the request-reply-fault message exchange pattern.

Figure 24-3 Request-Reply-Fault Message Exchange Pattern



Request-Callback Message Exchange Patterns

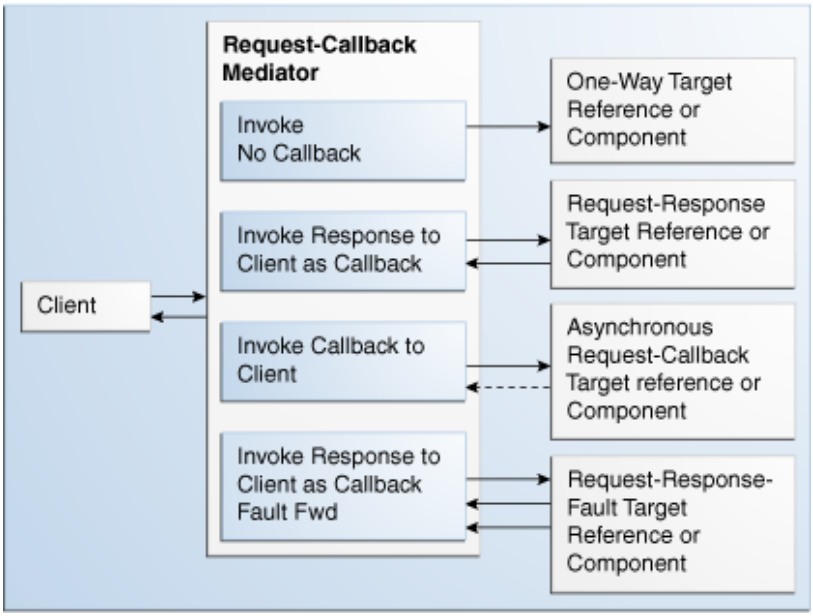
In a request-callback interaction, the Mediator is invoked and may send an asynchronous reply to the caller. Depending on the type of routing rule target, the responses, faults, and callbacks are handled as shown in Table 24-5:

Table 24-5 Response When 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.
Request Response	The response is sent back to the caller, as a callback, in a separate thread. You can create additional routing rules to forward the response to another target or event.
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. As above, you can create additional routing rules to forward the response 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 24-4 illustrates the request-callback message exchange pattern.

Figure 24-4 Request-Callback Message Exchange Pattern



Request-Reply-Callback Message Exchange Patterns

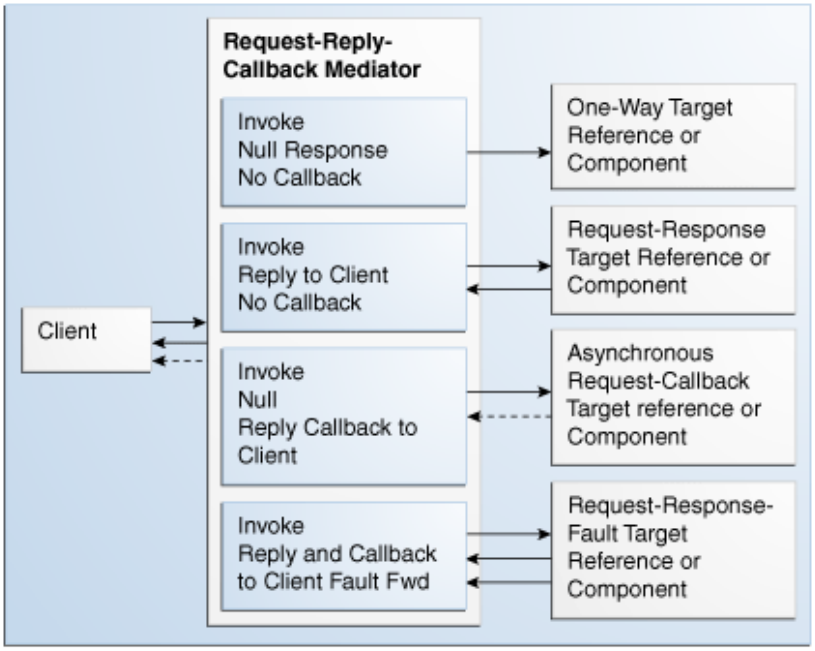
In a request-reply-callback interaction, the Mediator is invoked and sends a response and an asynchronous reply to the initial caller. Depending on the type of routing rule target, the responses, faults, and callbacks are handled as shown in Table 24-6:

Table 24-6 Response When 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. The fault is forwarded to another target or event.
Request Callback	There should be at least one sequential routing rule that returns a response. Mediator returns `null` when there is no response to be sent.
Request Response Callback	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 24-5 illustrates the request-reply-callback message exchange pattern.

Figure 24-5 Request-Reply-Callback Message Exchange Pattern



Request-Reply-Fault-Callback Message Exchange Patterns

In a request-reply-fault-callback interaction, the 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 24-7:

Table 24-7 Response to a Request Response Fault Callback Mediator

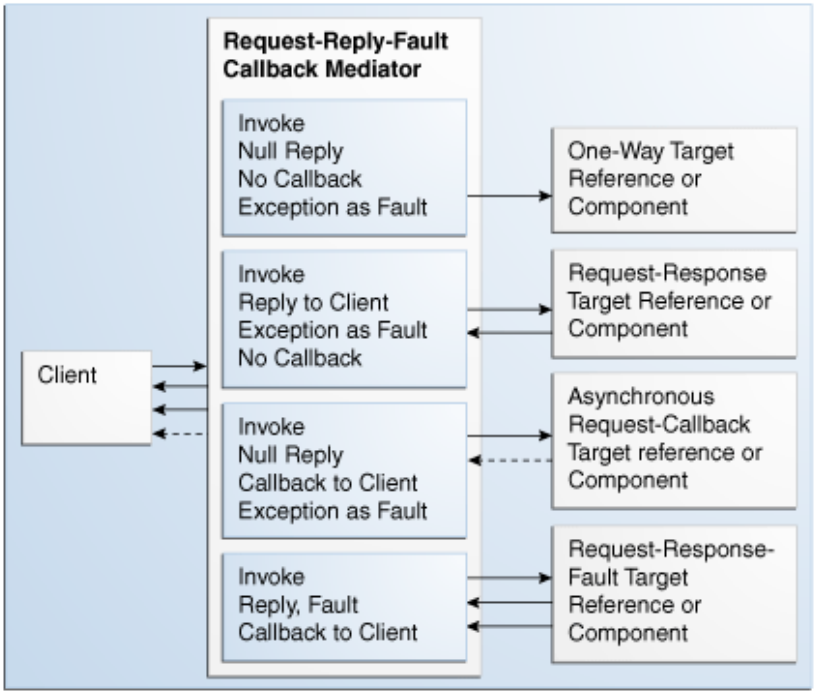
WSDL of the Routing Rule Target	Response
Request	There should be at least one sequential routing rule with a request-callback service and at least one sequential routing rule that returns a response. No callback or response is sent unless the required routing rules are defined.
Request Response	There should be at least one sequential routing rule with a request-callback service and at least one sequential routing rule that returns a response. No callback or response is sent unless the required routing rules are defined.
Request Response Fault	There should be at least one sequential routing rule with a request-callback service and at least one sequential routing rule that returns a response. No callback or response is sent unless the required routing rules are defined.
Request Callback	There should be at least one sequential routing rule that returns a response. Mediator returns `null` when there is no response to be sent.
Request Response Callback	The response and callback are sent back to the caller. Any exception in Mediator message processing may result in a fault.

Table 24-7 (Cont.) Response to a Request Response Fault Callback Mediator

WSDL of the Routing Rule Target	Response
Request Response Fault Callback	The response, fault, and callback are sent back to the caller.

Figure 24-6 illustrates the request-reply-fault-callback message exchange pattern.

Figure 24-6 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:

25

Getting Started with Oracle Business Rules

This chapter describes how to use a business rule service component to integrate a SOA composite application with . A business rule service component is also called a decision component. You can add business rules as part of a SOA composite application or as part of a BPEL process.

This chapter includes the following sections:

- Introduction to the Business Rule Service Component
- Overview of Rules Designer Editor Environment
- Introduction to Creating and Editing Business Rules
- Adding Business Rules to a BPEL Process
- Adding Business Rules to a SOA Composite Application
- Running Business Rules in a Composite Application
- Using Business Rules with Fact Types

For more examples of using , see *Designing Business Rules with Oracle Business Process Management*.

Note that some screen shots may reflect a previous version, however, the content is applicable.

Introduction to the Business Rule Service Component

A decision component, also called a business rule service component, supports use of 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 Creating Routing Rules .
- A decision component can be used with the Advanced Routing Rules in Human Workflow.

  For more information, see Associating Human Tasks with BPEL Processes.

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.

- Externalizing 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 data 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.

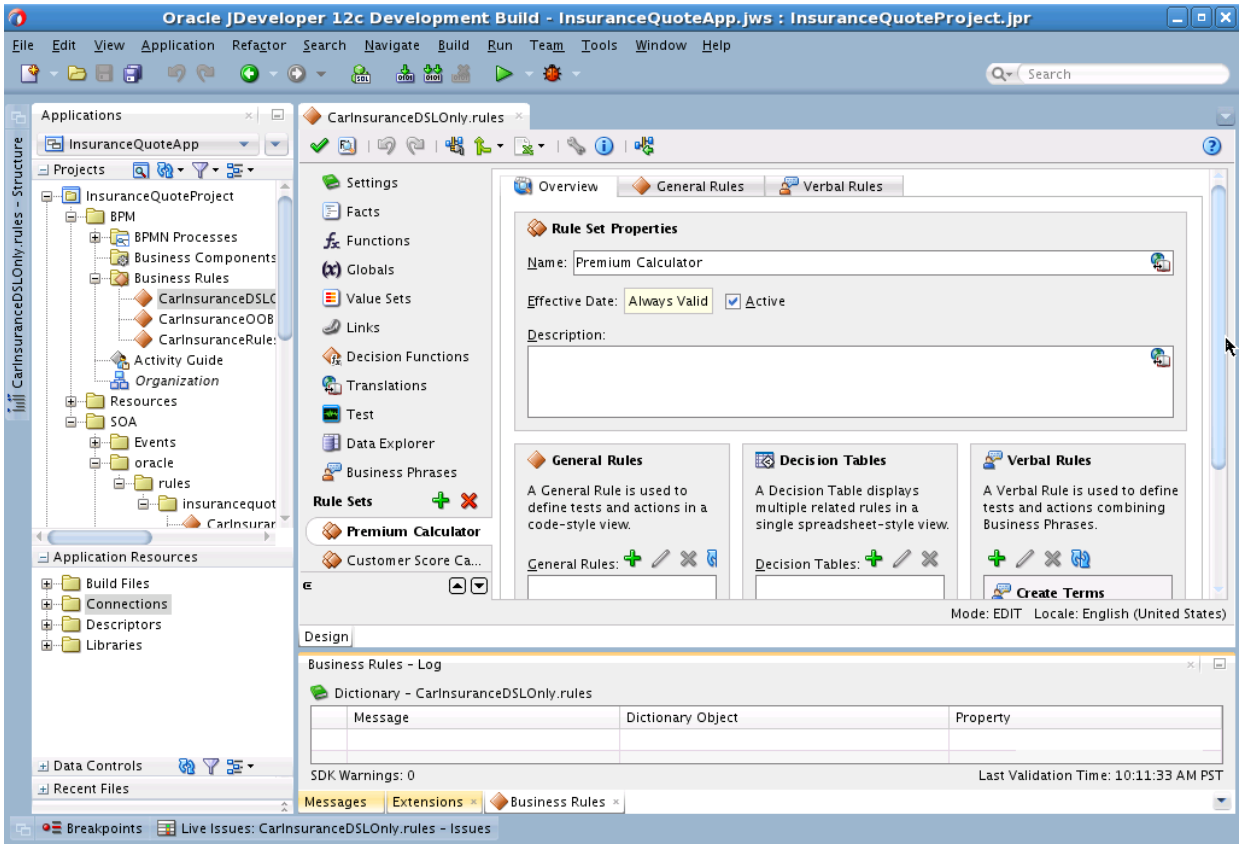
For more information about creating business rules in the Human Task editor of a human task component, see How to Specify Advanced Task Routing Using Business Rules.

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 25-1. These sections allow you to work with business rules in Oracle JDeveloper.

Figure 25-1 Rules Designer in Oracle JDeveloper



Note that a SOA installation does not have Verbal Rules or Business Phrases. This is BPM functionality.

Applications Window

The Applications window 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 Applications window.

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 Applications window, double-click a business rules dictionary file (a file with the **.rules** extension).

- Click the **Design** tab with a **.rules** file selected.

Table 25-1 describes where you can find information about working with a dictionary with Rules Designer.

Table 25-1 Rules Designer Navigation Areas Descriptions

Rules Designer Navigation Tab	Description
Facts	olink:ASRUG243Facts are the objects that rules reason on.
Functions	olink:ASRUG296A function, in Oracle Business Rules, refers to the standard mathematical functions.
Globals	olink:ASRUG277A global, in Oracle Business Rules, is similar to a public static variable in Java.
Value Sets	olink:ASRUG243A Value Set puts constraints on values or ranges of values for selection in a decision table.
Links	olink:ASRUG271Links are used to link to a dictionary in the same application or in another application.
Decision Functions	olink:ASRUG99955A decision Function is a function that is configured declaratively. It can be invoked by other components (BPEL, Task) to reason on inputs based on configured rulesets to arrive at outputs.
Translations	This helps you localize the rules and their artifacts.
Rulesets with Rules and s	A ruleset provides a unit of execution for rules and for decision tables. A decision table is a set of rules written in tabular form. Decision Tables provides additional functionality for rules that are grouped in the table (conflicts, completeness, and so on.).

For more information and descriptions for the Rules Designer navigation areas, see *Designing Business Rules with Oracle Business Process Management*.

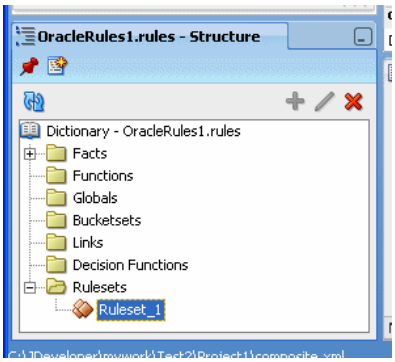
Structure Window

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 the element, including:

- Managing (creating, editing, refreshing, and deleting) elements such as facts, functions, globals, value sets, dictionary links, and decision functions
- Accessing rule sets, rules, and s

Figure 25-2 shows the Structure window.

Figure 25-2 Structure Window with Rules Designer Dictionary



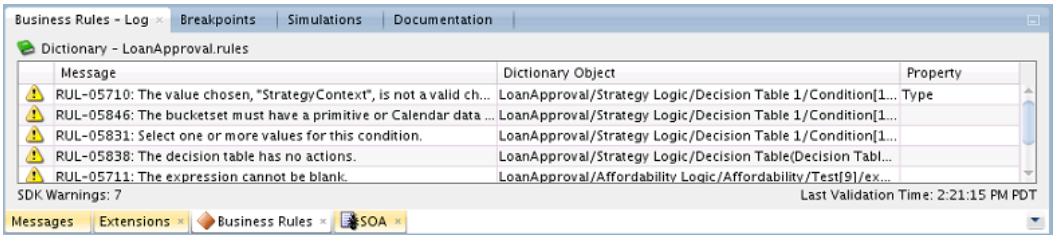
Business Rule Validation Log Window

Rules Designer displays the status of dictionary validation in the business rule validation log, as shown in Figure 25-3.

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 *Designing Business Rules with Oracle Business Process Management*.

Figure 25-3 Rules Designer Business Rule Validation Log



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.

How to Create Business Rules Components

You can add Business Rule components using the .

To create a Business Rule component:

1. Follow the instructions in Table 25-2 to start Oracle JDeveloper.

Table 25-2 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, drag a **Business Rule** service component from the Components window into the

   In a new application:

   a. From the Applications window, 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 25-3.

      Table 25-3 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 to appear.

3. Provide the required details. For more information on providing Inputs and Outputs and on using the **Import Dictionary** option with this dialog, see *Designing Business Rules with Oracle Business Process Management*.

4. Click **OK**.

Working with Business Rules in Rules Designer

When you are working with business rules Oracle JDeveloper displays Rules Designer.

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.

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.

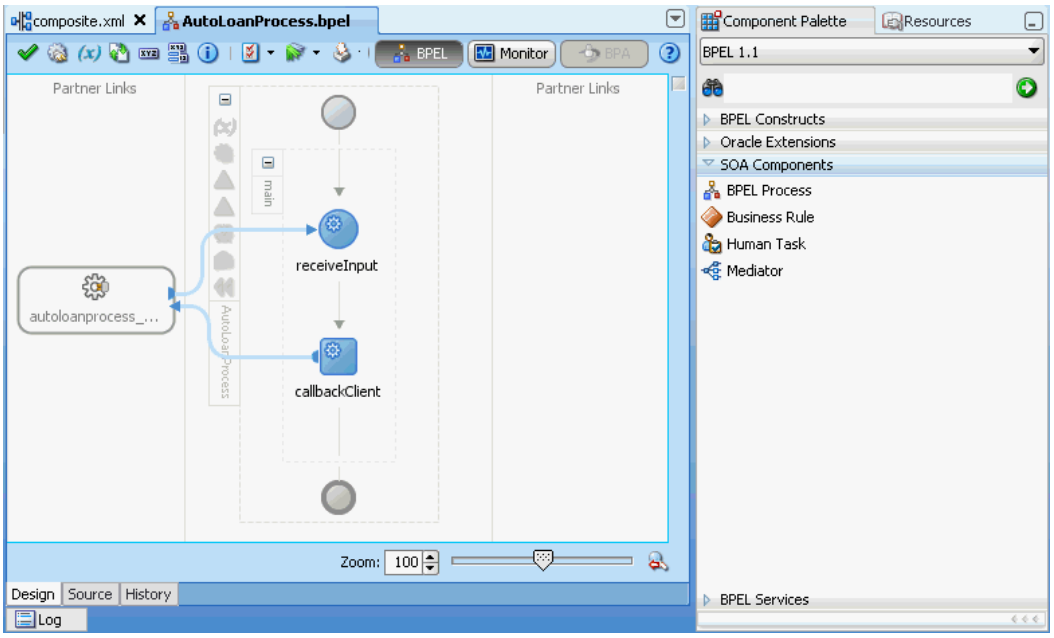
A business rule component enables you to execute business rules and make business decisions based on the rules. To create a business rule component, also called a decision component, you drag-and-drop a **Business Rule** from the Components window into the BPEL process.

To add a business rule to a BPEL process:

1. Create a BPEL process service component. For more information, see Introduction to the BPEL Process Service Component.

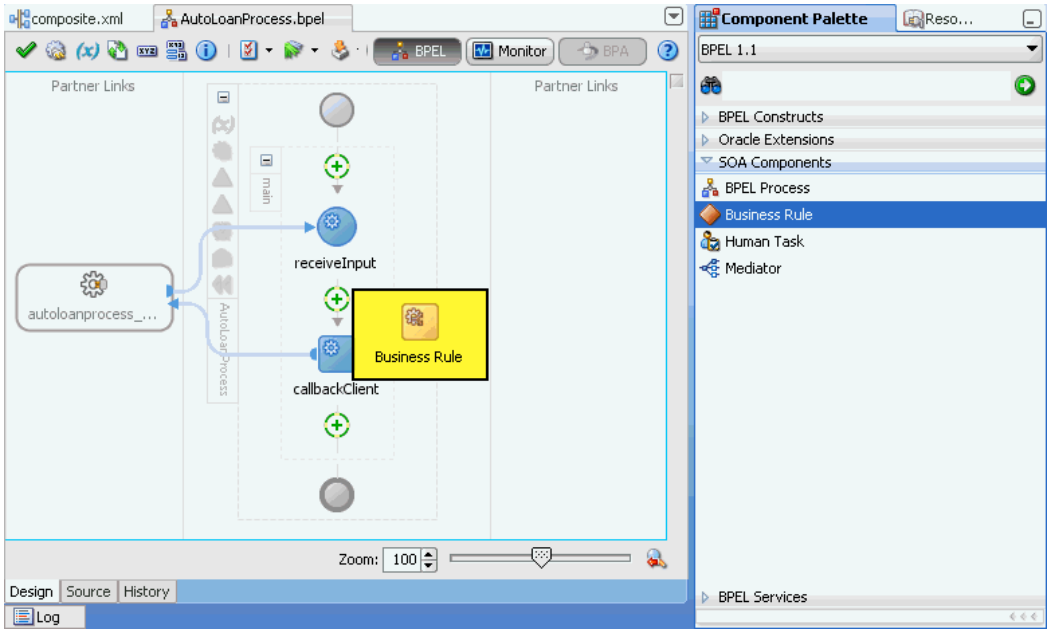
2. Expand the BPEL process by double-clicking the process item. For example, expand the BPEL process to view `receiveInput` and `callbackClient` as shown in Figure 25-4.

Figure 25-4 Adding A Business Rule to a BPEL Process



3. Select **Business Rule** from the SOA Components section of the Components window 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 25-5.

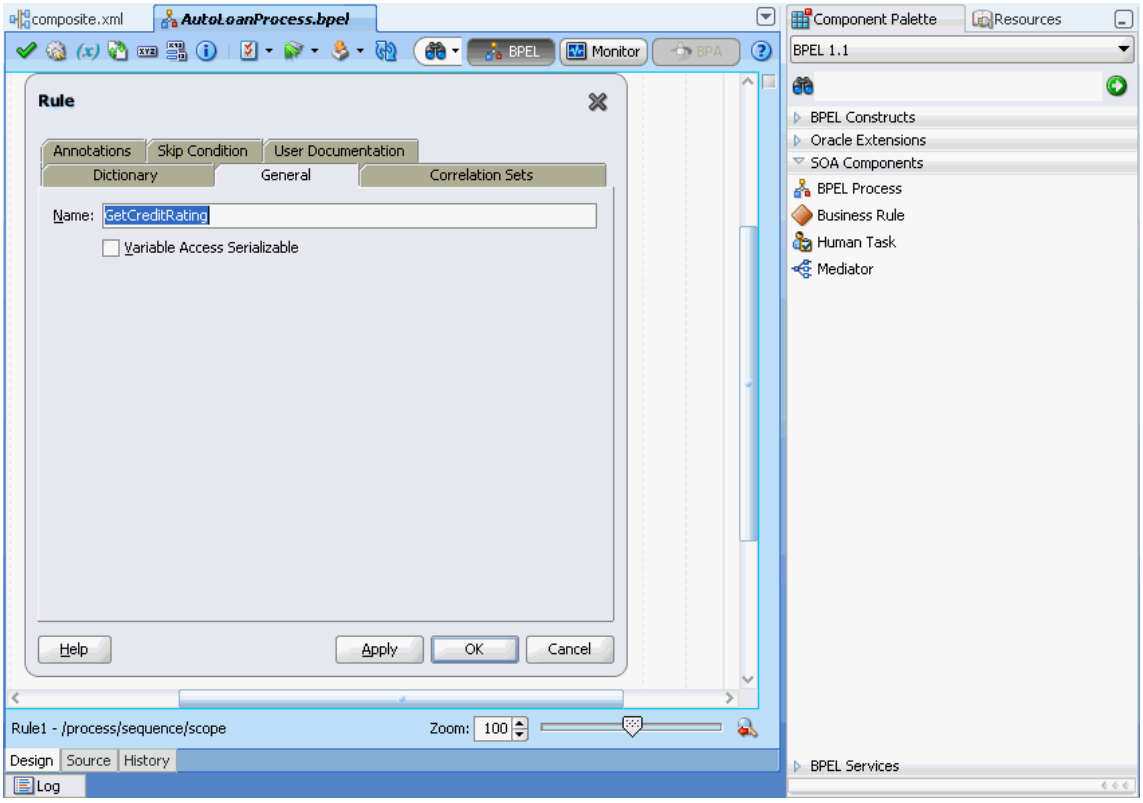
Figure 25-5 Drag-and-drop a Business Rule into a BPEL Process



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 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 25-6. If you previously created a dictionary, under the **Dictionary** tab, in the **Dictionary** field, select an existing dictionary.

Figure 25-6 Business Rule Added to Auto Loan BPEL Process



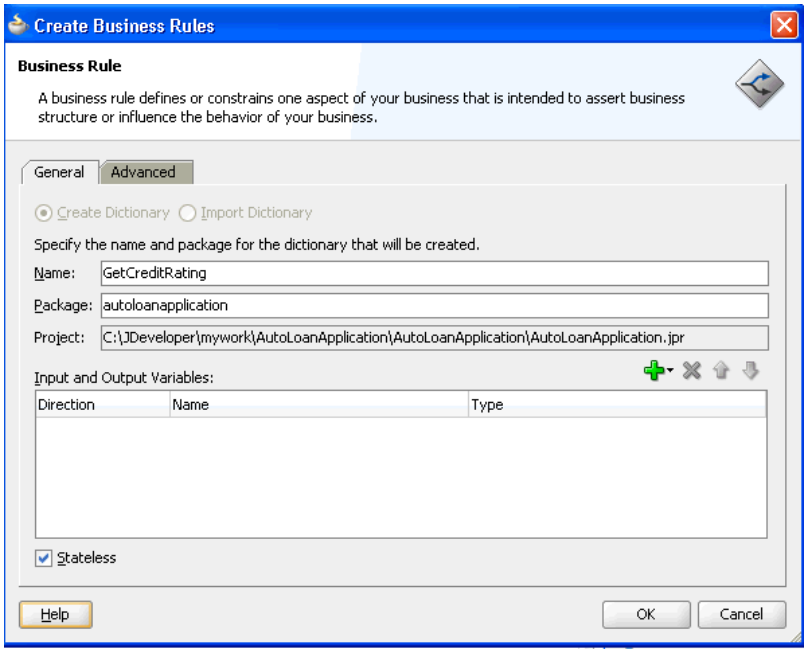
5. In the Business Rule area for the **Business Rule Dictionary**, click the **Create Dictionary** icon to display the Create Business Rules dialog.

6. In the Create Business Rules dialog you do the following:

   - Specify a name for the 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 dictionary. For example, enter `GetCreditRating`, as shown in Figure 25-7.

Figure 25-7 Adding GetCreditRating Business Rule Dictionary



How to Add Inputs for Business Rule

To add inputs for business rule:

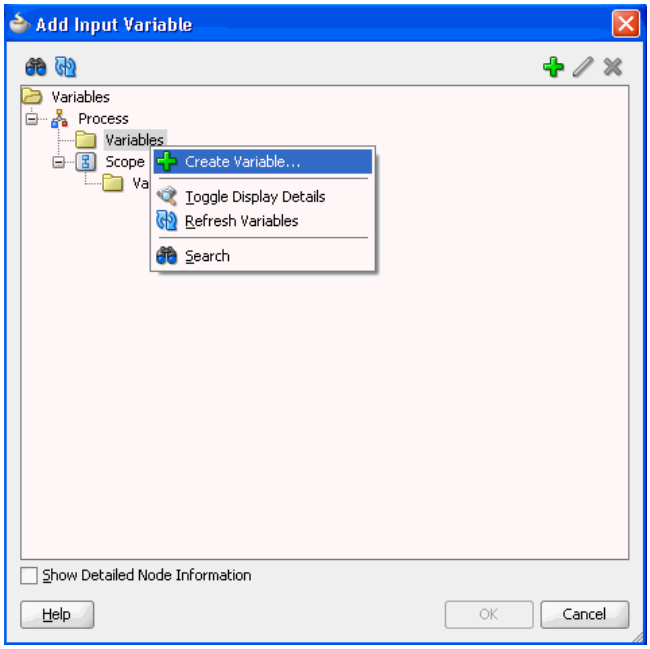
1. In the Create Business Rules dialog, from the menu next to the **Add** icon select **Add Input Variable...** to create the input variable.

   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 25-8.

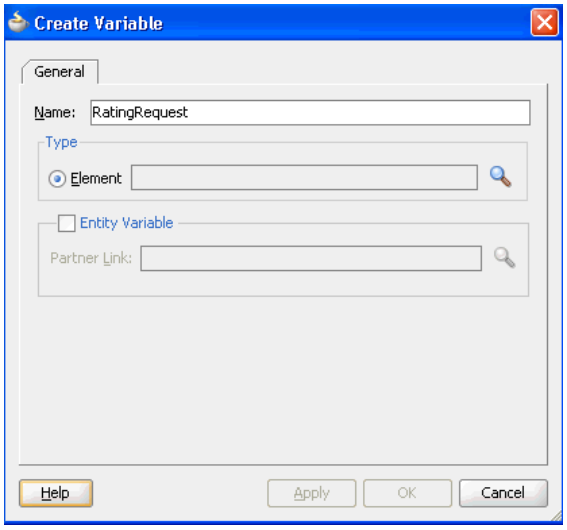
Figure 25-8 Add Input Variable



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 25-9.

Figure 25-9 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**.

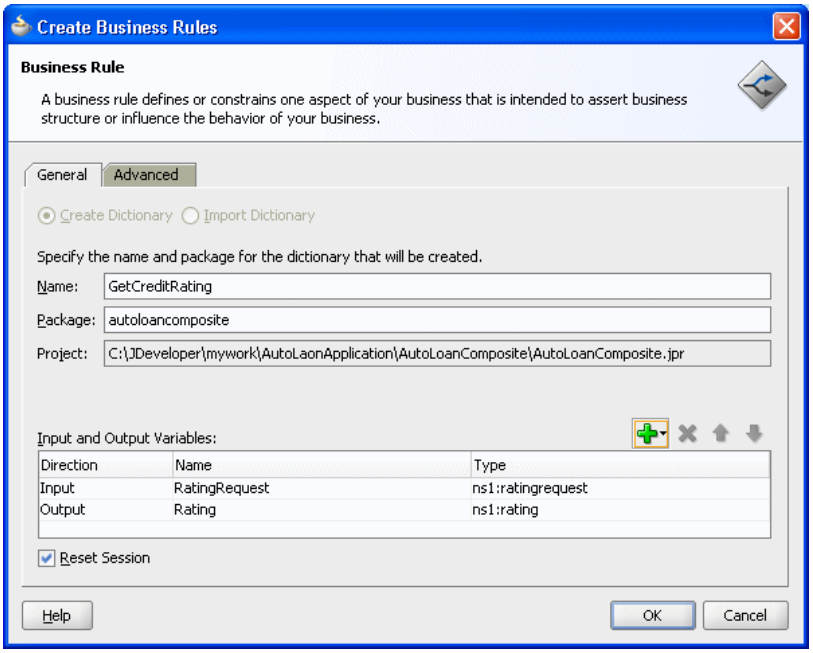
How to Add Outputs for Business Rule

To 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 25-10.

Figure 25-10 Create Business Rules Dialog with Input and Output Variables



How to Set Options and Create Decision Service and Business Rule Dictionary

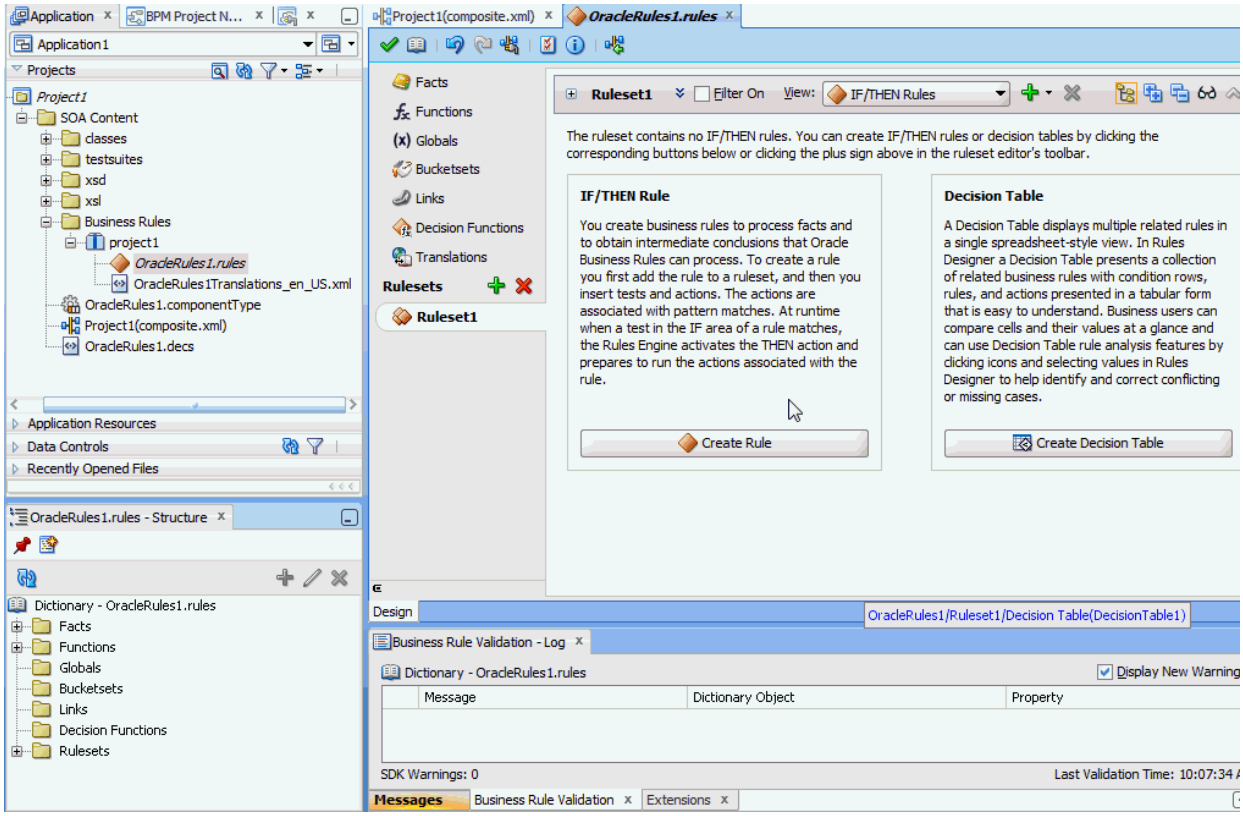
To create decision service and business rules dictionary:

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 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 25-11.

Figure 25-11 Rules Designer Canvas Where You Work with Business Rules



For information on Rules Designer, see *Designing Business Rules with Oracle Business Process Management*.

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 s that are evaluated using the . 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 s is exposed as a service with facts that are input and output, and the name of an 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 dictionary. For more information, see *Designing Business Rules with Oracle Business Process Management*.
- 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 *Designing Business Rules with Oracle Business Process Management*.

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 s. With an existing dictionary, you use the import option to specify a dictionary that may already contain the rules or s.
- 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.

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 with the inputs you supply, and returning the outputs with results. The decision service provides access to Engine at runtime as a web service. For more information, see *Designing Business Rules with Oracle Business Process Management*.

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** check box in the Create Business Rules dialog provides support for these two modes of operation.

By default the **Reset Session** check box 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 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.

Adding Business Rules to a SOA Composite Application

To work with 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 dictionary

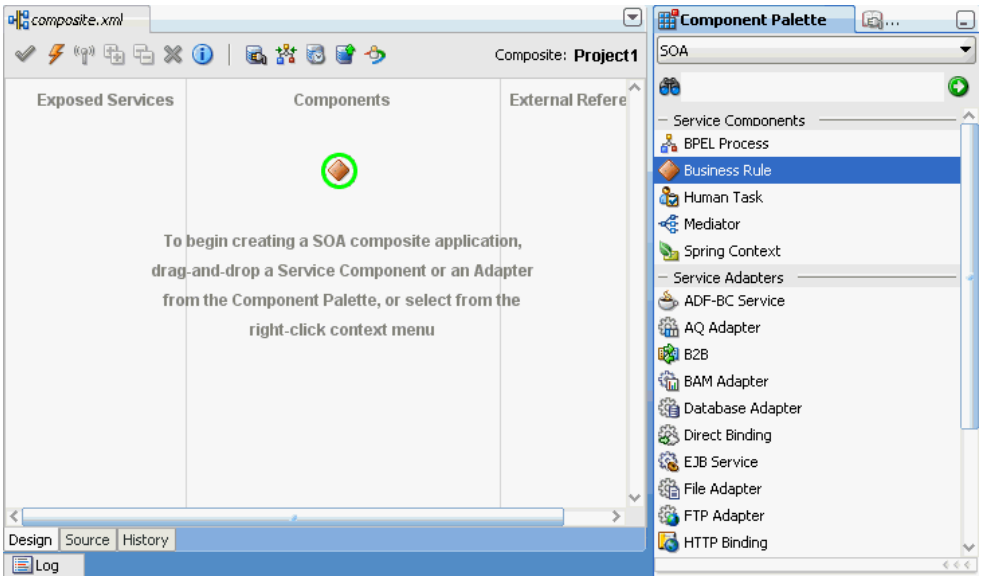
How to Add Business Rules to a SOA Composite Application

To work with 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:

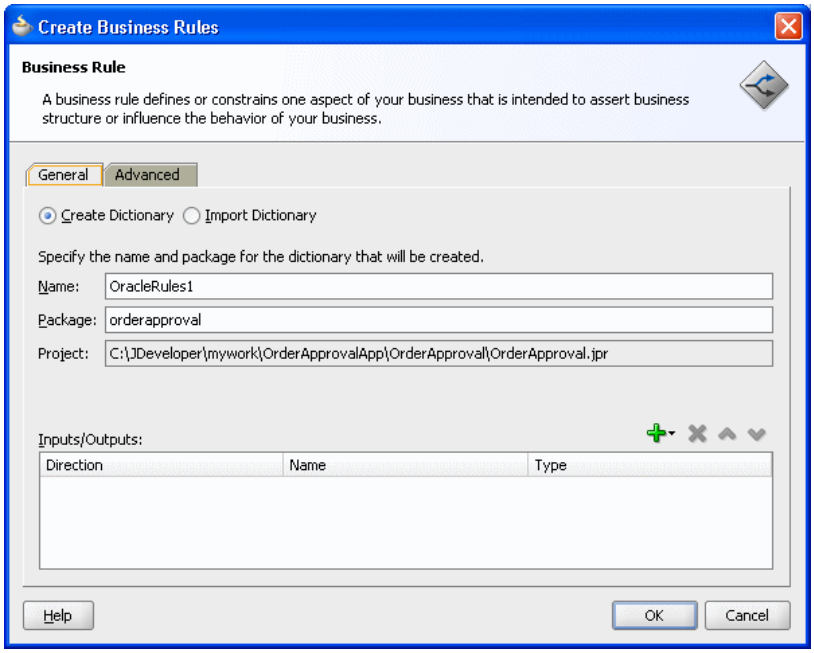
1. Create a SOA application and project. For more information, see How to Create a SOA Application and Project. For a SOA composite using business rules, pick the required technologies for your application. For example, you may need the following for a SOA application with business rules: ADF Business Components, Java, and XML. You move these items to the **Selected** area on the Project Technologies tab.
2. In the Applications window, 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 Components window, 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 25-12.

Figure 25-12 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 25-13.

Figure 25-13 Adding Business Rules to a SOA Composite and Creating a Dictionary

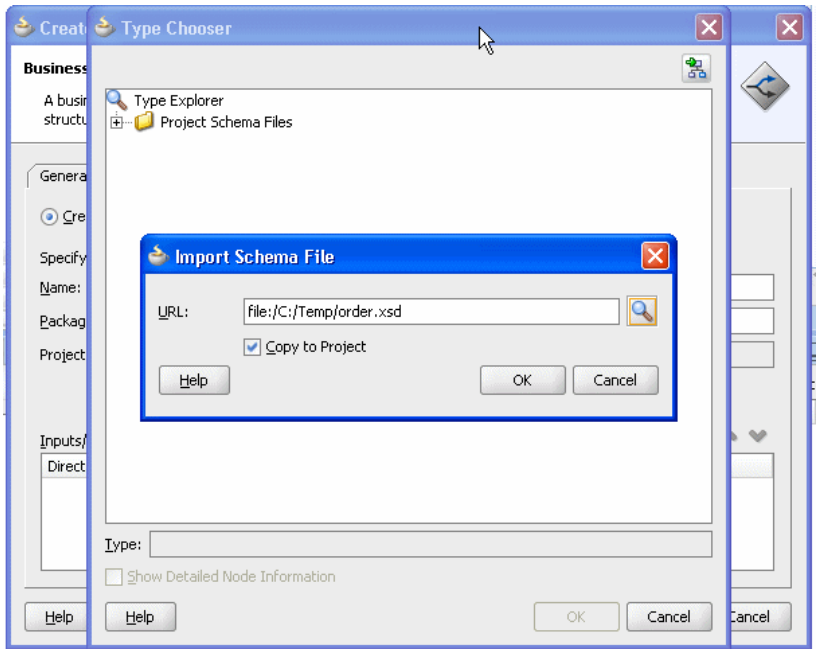


How to Add Inputs to a Business Rule

To add inputs to a business rule:

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 25-14.

Figure 25-14 Importing Schema for Input to Business Rules



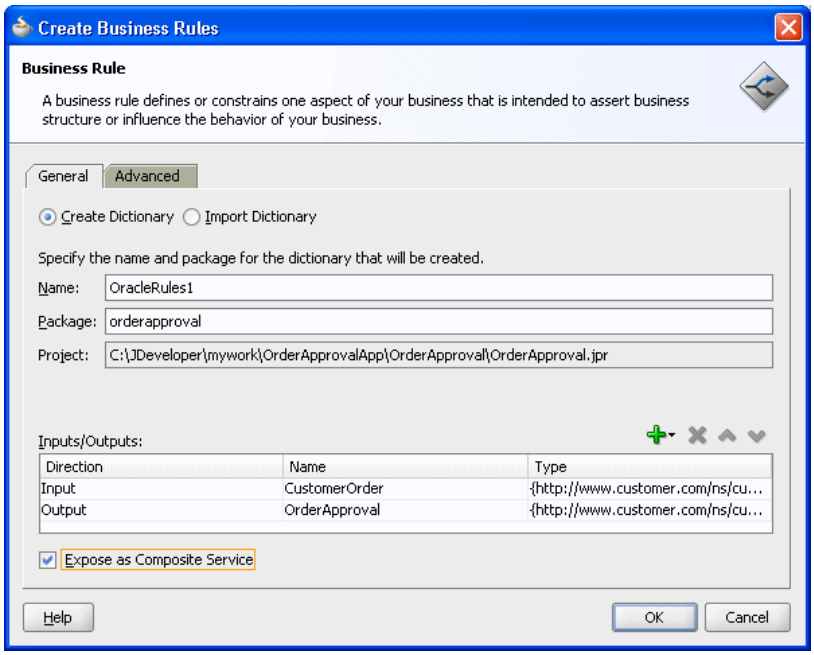
7. In the Import Schema File dialog, click **OK**.
8. In the Localize Files dialog, click **OK**.
9. Use the Type Chooser dialog navigator to locate and select the input from the schema and click **OK**. For example, select the `CustomerOrder` element as the input.

How to Add Output to a Business Rule

To add outputs to a business rule:

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 25-15.

Figure 25-15 Create Business Rules Dialog with Input and Output

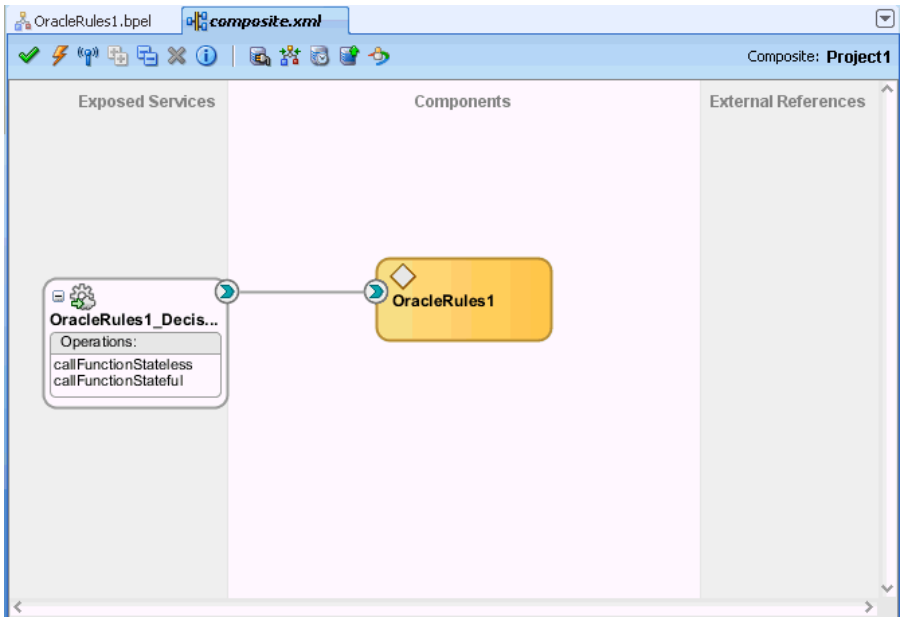


How to Set Options and Create Decision Service and Business Rules Dictionary

To create decision service and business rules dictionary:

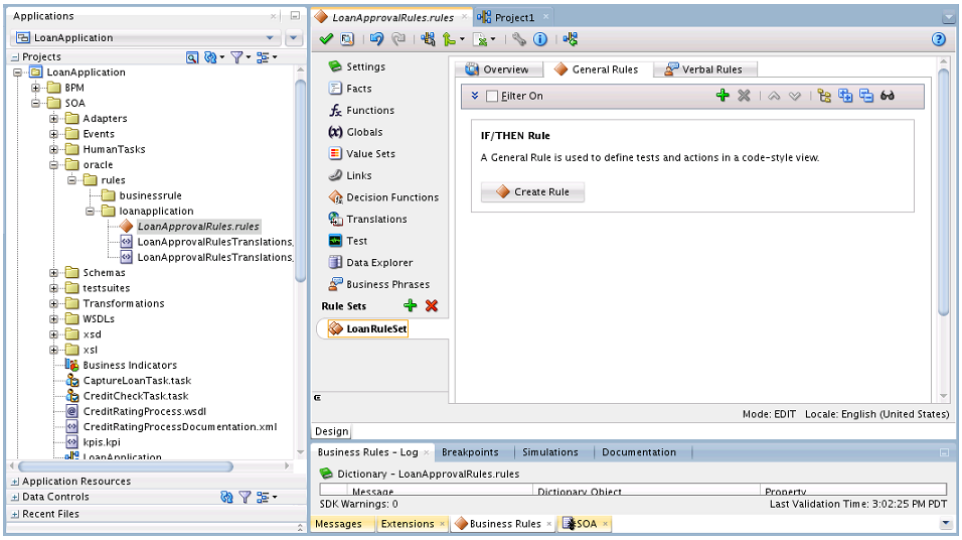
1. In the Create Business Rules dialog, select **Expose as Composite Service**.
2. If you do not want to use the default service name, then select the **Advanced** tab and in the **Service Name** field enter the service name.
3. In the Create Business Rules dialog, click **OK**. This creates the Business Rule component, also called a decision component, and Oracle JDeveloper shows the Business Rule component in the canvas workspace as shown in Figure 25-16.

Figure 25-16 Business Rule Component in SOA Composite



4. Double-click the decision component to open Rules Designer, as shown in Figure 25-17. 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 25-17 Rules Designer Showing New Dictionary for SOA Composite Application



For information on Rules Designer, see *Designing Business Rules with Oracle Business Process Management*.

Note that a SOA installation does not have Verbal Rules or Business Phrases. This is BPM functionality.

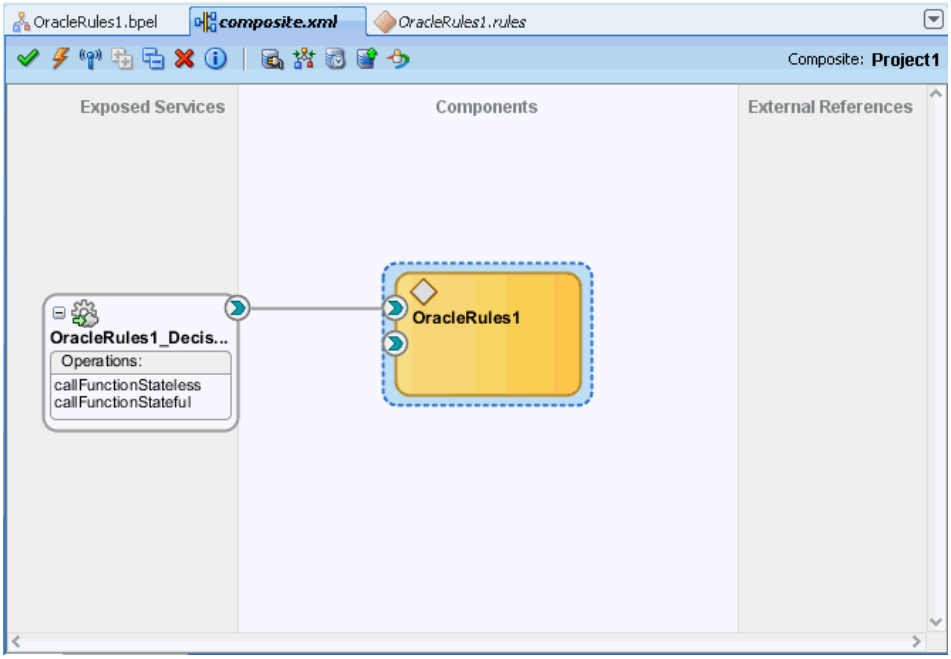
How to Select and Modify a Decision Function in a Business Rule Component

You can specify one or more decision functions as inputs for calling 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 dictionary.

To specify a decision function in a composite application:

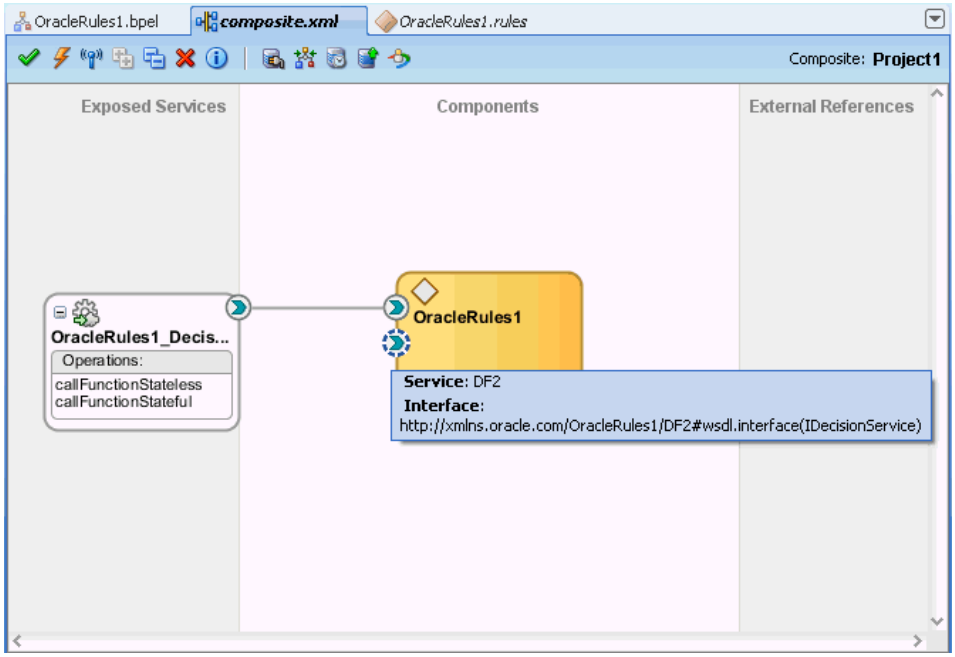
1. Add a decision function to the dictionary. For more information, see *Designing Business Rules with Oracle Business Process Management*.
2. Add a Business Rule component to the composite application. For more information, see How to Add Business Rules to a SOA Composite Application.
3. Select a business rule component, as shown in Figure 25-18.

Figure 25-18 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 25-19.

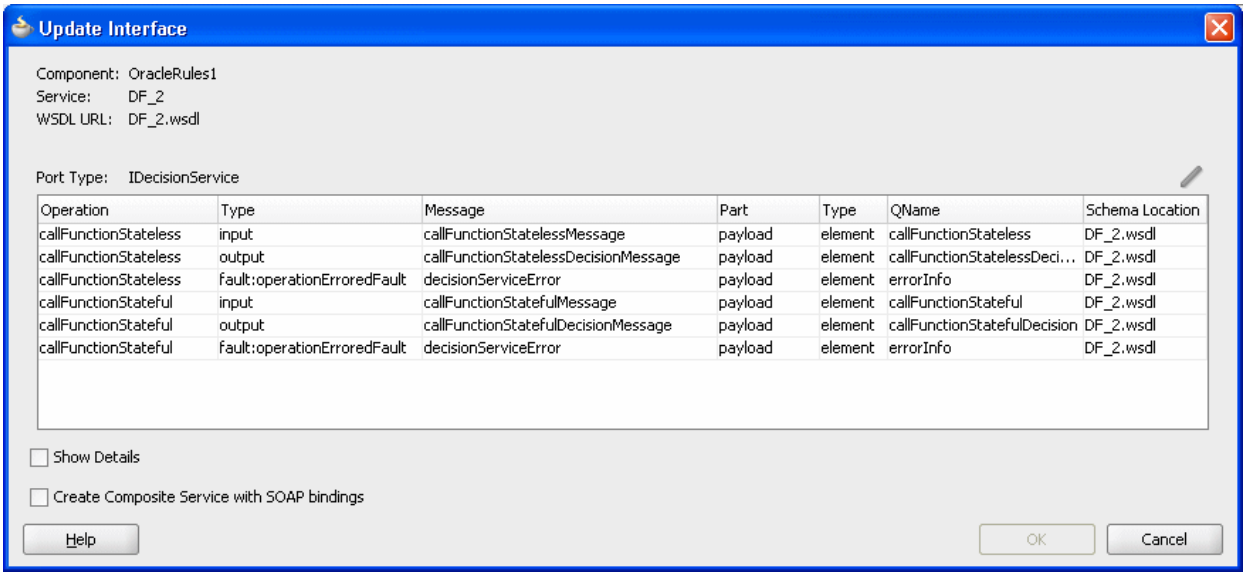
Figure 25-19 Selecting a Decision Function Port in a Business Rule Component



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 25-20.

Figure 25-20 Update Interface Dialog for a Decision Function in a Business Rule Decision Port



Running Business Rules in a Composite Application

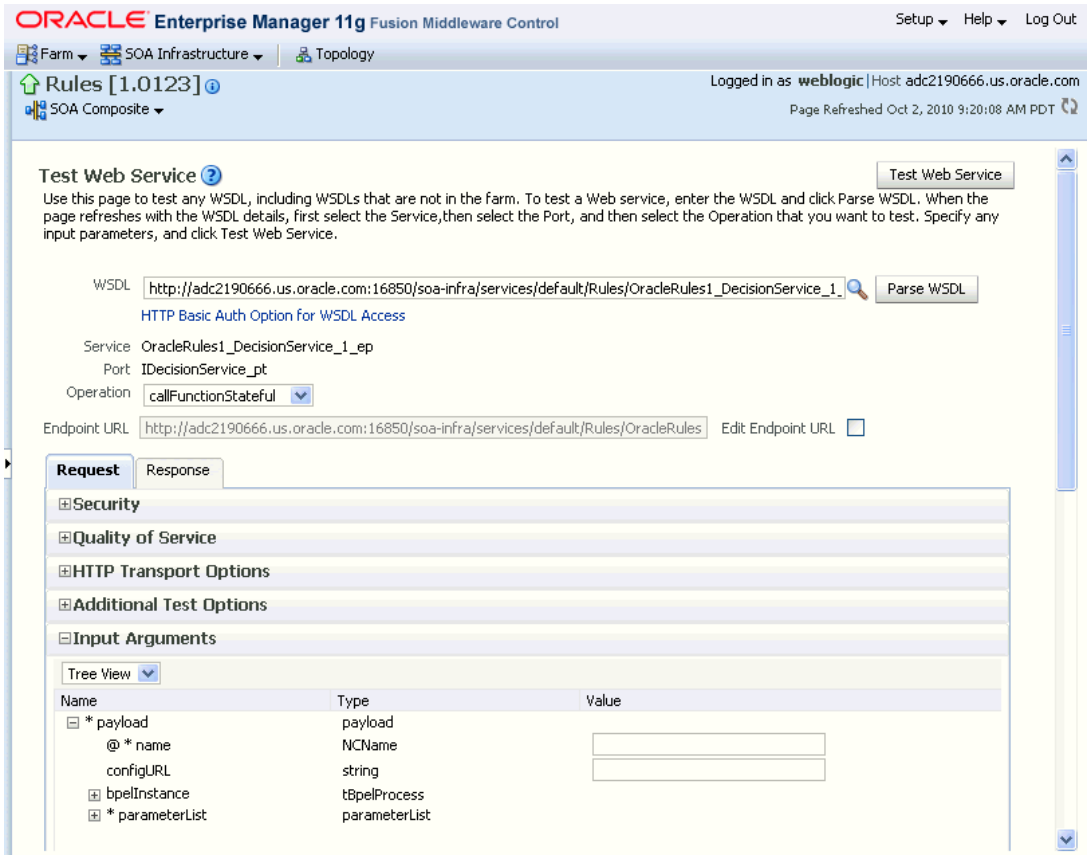
You run business rules as part of a decision component within a SOA composite application. The business rules are executed by the Business Rule Service Engine. You can use Oracle Enterprise Manager Fusion Middleware Control to monitor the Business Rule Service Engine and to test a SOA composite application that includes a

decision component. For more information, see *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

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 25-21.

Figure 25-21 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 25-22.

Figure 25-22 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 is not possible to invoke the standalone decision service with just the payload and endpoint details.

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 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 decision functions in the `DecisionPointDictionary` and classes in the 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 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 -specific samples available with the Oracle SOA Suite samples.

26

Using Declarative Components and Task Flows

This chapter describes how to use different declarative components and task flows to develop high-performance, interactive, and multitiered applications that are also easy to maintain. It describes how to use the Oracle Business Rules Editor declarative component and the Oracle Business Rules Dictionary Editor declarative component and task flow. It also describes how to localize the ADF-based web application.

This chapter includes the following sections:

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.

Introduction to 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.

Using the Oracle Business Rules Editor Component

The 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. The Rules Editor uses the Rules SDK2 API to create and edit rules.

Note:

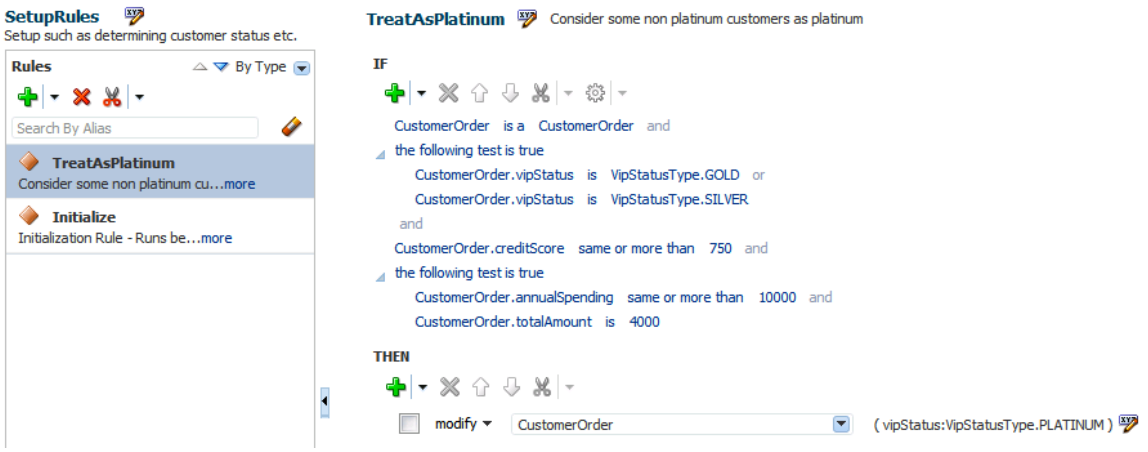
You should not confuse the Rules Editor with the Rules Dictionary Editor. The Rules Editor is used to edit rules inside a specified ruleset. In fact, the Rules Editor is embedded within the Rules Dictionary Editor. For more information about the Rules Dictionary Editor, see Introduction to the Oracle Business Rules Dictionary Editor Declarative Component.

Using the Rules Editor, you can create, delete and edit the general rules, verbal rules, and decision tables that are part of a single ruleset. You are required 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:

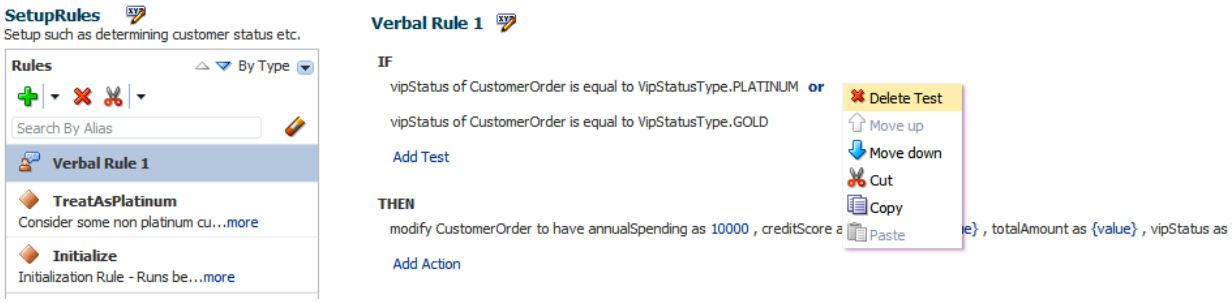
- Creates, updates, and deletes:
  - Rules in a ruleset, as shown in Figure 26-1.
  - Simple tests or conditions in a rule, as shown in the IF area.
  - Actions in a rule, as shown in the THEN area.

Figure 26-1 General Rules in a Ruleset



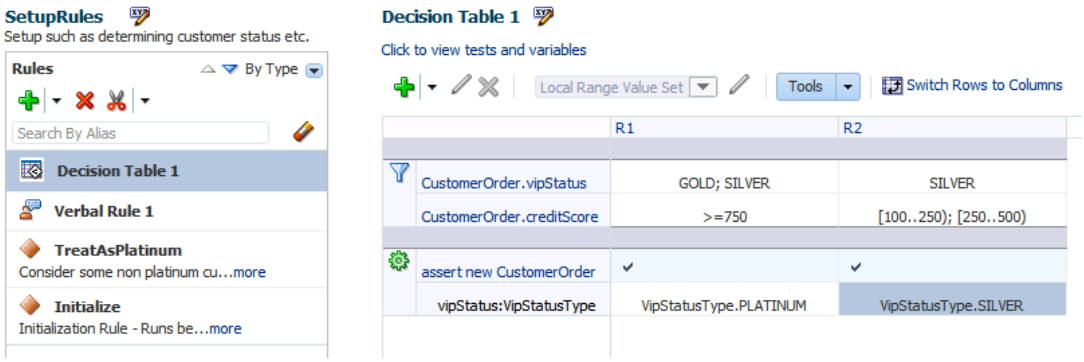
– Verbal rules, as shown in Figure 26-2.

Figure 26-2 Verbal Rules in a Ruleset



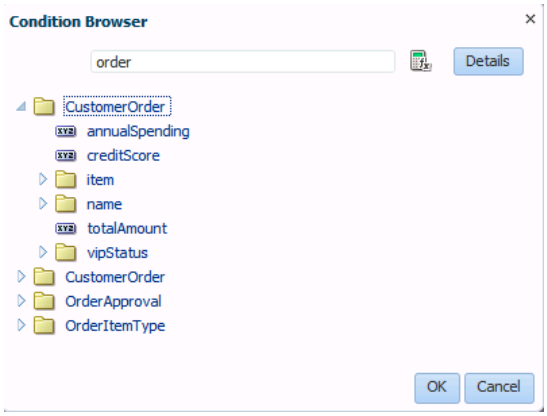
– Decision tables, as shown in Figure 26-3.

Figure 26-3 Decision Table



- 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 26-4.

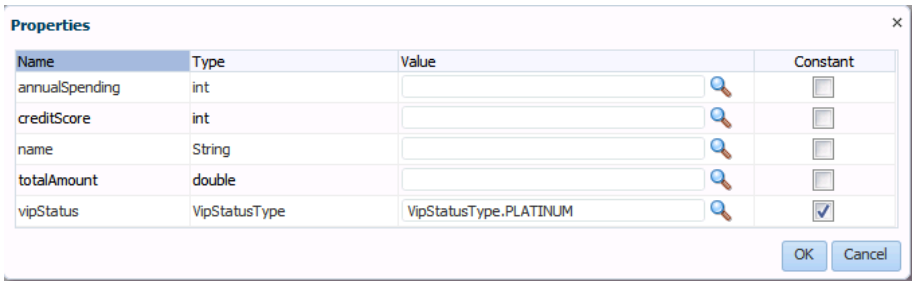
Figure 26-4 Condition Browser



- Provides a Date Browser for selecting date types.
- Provides a Right Operand browser to handle multiple right-hand side expressions.
- Provides support for nested rules.

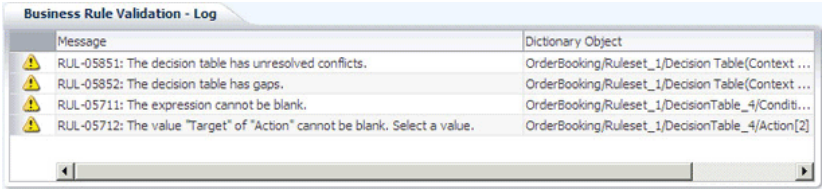
- Provides the Properties browser for editing properties of a rule action, as shown in Figure 26-5.

Figure 26-5 Properties Browser



- Provides an Expression Builder window to build custom expressions.
- Provides a Validation panel to manage error messages, as shown in Figure 26-6.

Figure 26-6 Validation Panel to Manage Error Messages



Note:

After all the edits are done, the component user is responsible for saving the ruleset.

How to Create and Run a Sample Application by Using the Rules Editor Component

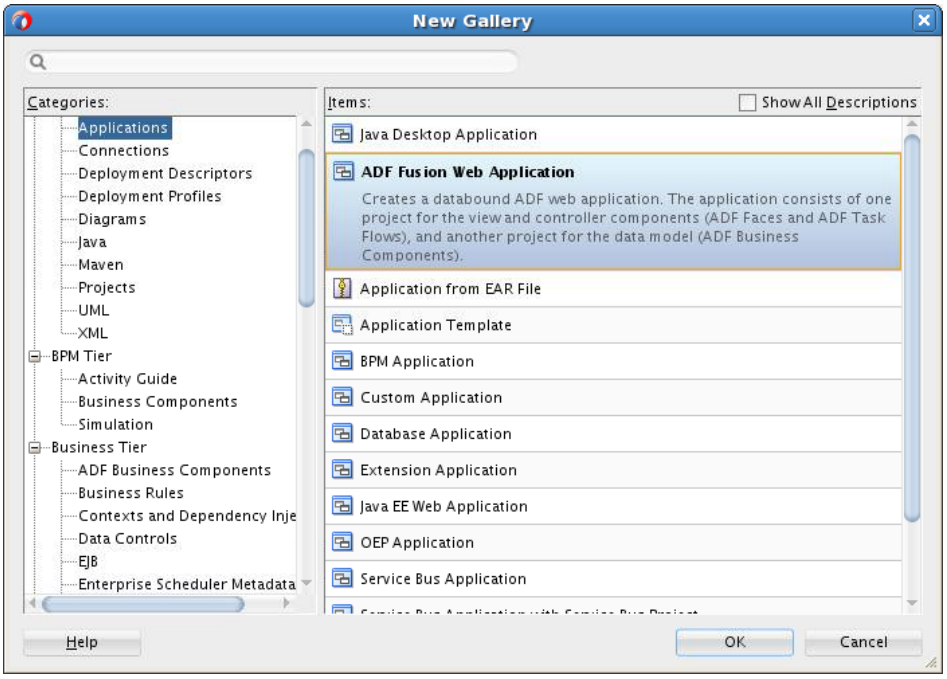
This section lists the steps for creating and running a sample application by using the Rules Editor component.

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:

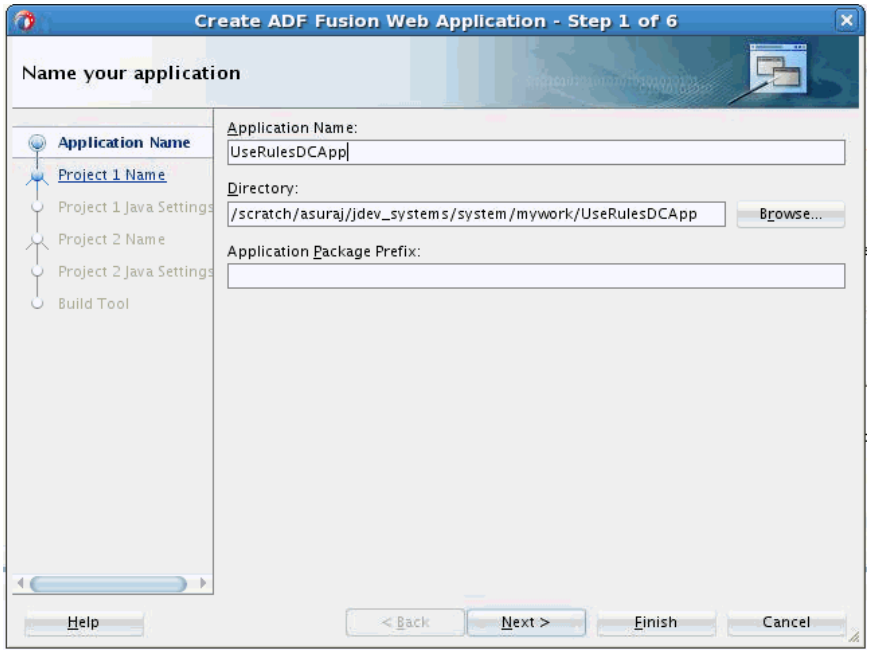
1. Open Oracle JDeveloper.
2. From the **File** menu, select **New**.
3. Select **ADF Fusion Web Application** to create a new application as shown in Figure 26-7.

Figure 26-7 Creating Fusion Web Application



4. Enter a name for the application in the **Application Name** field, for example, `useRulesDCApp`, and click **Next** as shown in Figure 26-8.

Figure 26-8 Creating a Generic Application



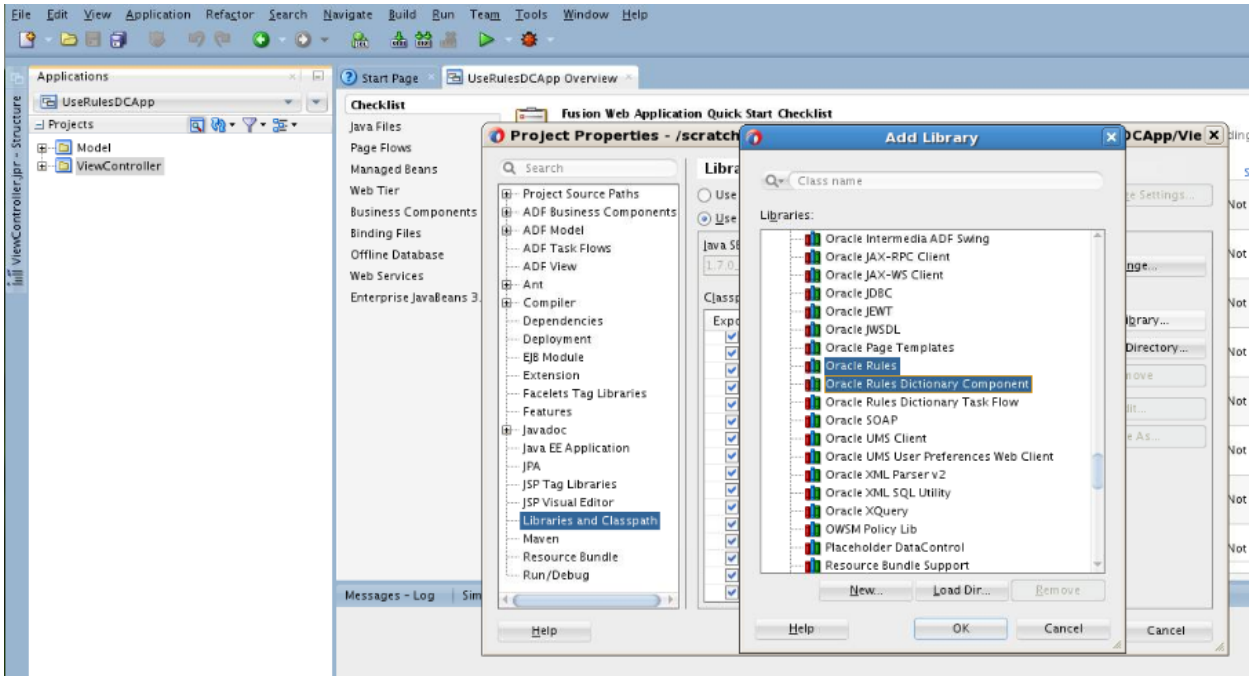
5. Use the default for everything else.
6. Click Finish.
7. Right click **ViewController** project and select **Project Properties**.
8. Select **Libraries and Classpath** from the menu on the left.

9. **a.** Click the **Add Library** button.

   b. Select **Oracle Rules** and **Oracle Rules Dictionary Component** from the Extension List and click **OK**. This adds the Rules SDK and the Rules ADF component tag libraries to the project as shown in Figure 26-9.

   c. Click **OK** once more to come out of Project Properties.

Figure 26-9 Adding Library



Note:

If the 'Oracle Rules' and 'Oracle Rules Dictionary Component' do not show up in the 'Extension' List, open a SOA/BPM project within jDeveloper to load the required libraries.

---

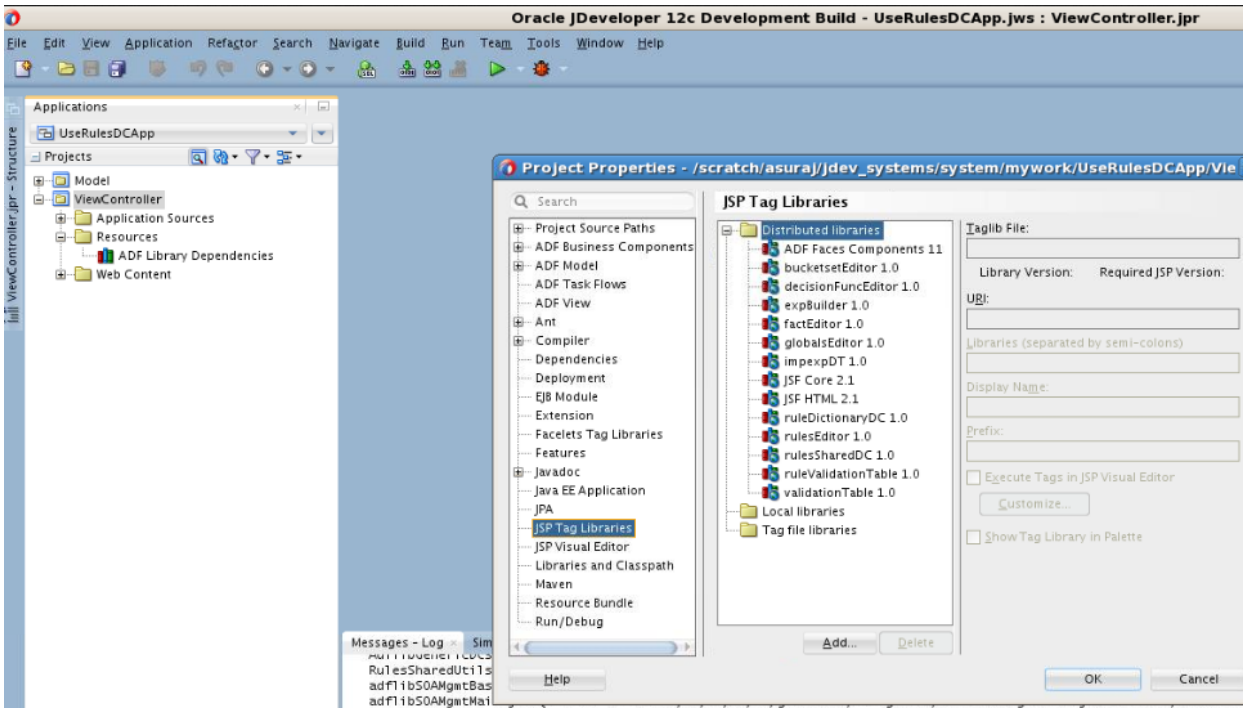
10. Click **Save All** to save the project.

11. Check to make sure all the required tag libraries are added.

12. **a.** Right click **ViewController project** and select **Project Properties.**

    b. Select **JSP Tag Libraries** from the menu on the left and check if all the tag libraries are added Figure 26-10.

Figure 26-10 Checking the Required Tag Libraries



How to Create the RuleSetModel Object

The Rules Editor component requires a `oracle.bpel.rulesdc.model.impl.RuleSetModel` object.

To create the RuleSetModel object:

1. Create a Java Class e.g. 'SomeBean.java' in your project.
2. Open Oracle JDeveloper.
3. From the **File** menu, select **New** and create a **Java Class**.
4. In SomeBean.java provide a method that returns the RuleSetModel object. You must specify the location/path of the rules file.The following is a sample of the `SomeBean.java` file:

```
package view;import java.io.BufferedReader;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.Serializable;

import java.net.MalformedURLException;
import java.net.URL;

import oracle.bpel.rulesdc.model.decisiontable.impl.DecisionTablePrefsImpl;
import oracle.bpel.rulesdc.model.decisiontable.interfaces.DecisionTablePrefs;
import oracle.bpel.rulesdc.model.impl.IfThenPreferencesImpl;
import oracle.bpel.rulesdc.model.impl.RuleSetModel;
import oracle.bpel.rulesdc.model.interfaces.IfThenPreferences;
import oracle.bpel.rulessharedutils.impl.RulesSharedUtils;
```

```
import oracle.rules.sdk2.decisionpoint.DecisionPointDictionaryFinder;
import oracle.rules.sdk2.dictionary.DictionaryFinder;
import oracle.rules.sdk2.dictionary.RuleDictionary;
import oracle.rules.sdk2.exception.SDKException;
import oracle.rules.sdk2.ruleset.RuleSet;
import oracle.rules.sdk2.ruleset.RuleSetTable;

public class SomeBean {
    //on windows
    private static final String RULES_FILE1 =
 "file:///D:/scratch/asuraj/system_MAIN/rules_files/ApprovalRules.rules";
    /*
     * on linux
    private static final String RULES_FILE1 =
 "file:////scratch/asuraj/backup/rules_files/ApprovalRules.rules";
    */
    private RuleSetModel ruleSetModel = null;

    private boolean viewOnly = true;
    private DecisionTablePrefs dtPrefs;

    private IfThenPreferences ifThenPrefs;

    public SomeBean() {
        super();
    }

    public RuleSetModel getRuleSetModel() {
        if (ruleSetModel != null)
            return ruleSetModel;
        ruleSetModel = new RuleSetModel(getRuleSet());
        System.out.println("ruleSetModel = " + ruleSetModel);
        return ruleSetModel;
    }

    public RuleSet getRuleSet() {

        RuleDictionary dict =
            openRulesDict(RULES_FILE1, new DecisionPointDictionaryFinder());
        if (dict == null)
            return null;

        RuleSetTable ruleSetTable = dict.getRuleSetTable();
        if (ruleSetTable == null || ruleSetTable.isEmpty())
            return null;

        return ruleSetTable.get(0);
    }

    public void saveDictionary() {

        RuleDictionary dict = null;
        String rulesFile = null;

        if (this.ruleSetModel == null)
            return;
        dict = this.ruleSetModel.getRuleSet().getDictionary();


        if (dict == null)
            return;
```

```
        if (dict.isModified())
            RulesSharedUtils.updateDictionary(dict);
        if (!dict.isTransactionInProgress())
            saveDictionary(dict, RULES_FILE1);
    }

    public void validate() {
        if (this.ruleSetModel == null)
            return;

        this.ruleSetModel.validate();
    }

    //utility methods

    public static RuleDictionary openRulesDict(String fileName,
DictionaryFinder finder) {
        URL url = null;
        try {
            url = new URL(fileName);
        } catch (MalformedURLException e) {
            System.err.println(e);
            return null;
        }
        RuleDictionary dict = null;

        try {
            dict = readFromDisk(url, finder);
        } catch (Exception e) {
            System.err.println(e);
            return null;
        }
        return dict;
    }

    public static RuleDictionary readFromDisk(URL dictURL, DictionaryFinder
finder) {
        BufferedReader buf = null;
        try {
            buf = new BufferedReader(new
InputStreamReader(dictURL.openStream(), "UTF-8"));
            return RuleDictionary.readDictionary(buf, finder);
        } catch (SDKException e) {
            System.err.println(e);
        } catch (IOException e) {
            System.err.println(e);
        } finally {
            if (buf != null)
                try {
                    buf.close();
                } catch (IOException e) {
                    System.err.println(e);
                }
        }

        return null;
    }

    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");

        try {
            writeToDisk(dict, new URL(ruleFileName));
        } catch (MalformedURLException e) {
            System.err.println(e);
            return false;
        } catch (Exception e) {
            System.err.println(e);
            return false;
        }
        return true;
    }

    public static void writeToDisk(RuleDictionary dic, URL dictURL) {
        OutputStreamWriter writer = null;
        try {
            writer = new OutputStreamWriter(new
FileOutputStream(dictURL.getPath()), "UTF-8");
            dic.writeDictionary(writer);
        } catch (IOException e) {
            System.err.println(e);
        } catch (SDKException e) {
            System.err.println(e);
        } finally {
            if (writer != null)
                try {
                    writer.close();
                } catch (IOException e) {
                    System.err.println(e);
                }
        }
    }

    public void toggleMode() {
        viewOnly = !viewOnly;
    }

    public boolean isViewOnly() {
        return viewOnly;
    }

    public DecisionTablePrefs getDtPreferences() {
        if (dtPrefs == null)
            dtPrefs = new DTPreferences();
        return dtPrefs;
    }

    public IfThenPreferences getIfThenPreferences() {
        if (ifThenPrefs == null)
            ifThenPrefs = new MyIfThenPrefs();
        return ifThenPrefs;
    }
    public class MyIfThenPrefs extends IfThenPreferencesImpl implements
Serializable {
```

```
        @Override
        public boolean isGenericAction() {
            return true;
        }

        @Override
        public boolean isGenericCondition() {
            return true;
        }
    }

    public class DTPreferences extends DecisionTablePrefsImpl implements
 Serializable {

        @Override
        public boolean isShowDTButtons() {
            return true;
        }
    }
}
```

5. Point to SomeBean.java in adfc-config.xml with Bean Name "someBean" and a "session" scope. Example adfc-config.xml:

```
<?xml version="1.0" encoding="UTF-8" ?>
<adfc-config xmlns="http://xmlns.oracle.com/adf/controller" version="1.2">
  <managed-bean id="__1">
    <managed-bean-name>someBean</managed-bean-name>
    <managed-bean-class>view.SomeBean</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
  </managed-bean>
</adfc-config>
```

6. The ADF/JSF framework makes calls to SomeBean.java multiple times to render the UI. For instance, someBean.ruleSetModel is called many times. So it is more efficient to create the ruleSetModel once and cache it and return it each time instead of recreating it.

How to Create the .jspx File

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

To create the .jspx file to include the Rules Editor Component tag:

1. Open Oracle JDeveloper.
2. From the **File** menu, select **New** and then select **JSF**.
3. Select **JSF Page** and click OK.
4. Select Document Type as **JSP XML**.
5. Enter `rulesEditor.jspx` as file name. Click **OK**.
6. The **RulesEditor** is visible in the component window in jDeveloper.
7. Select **RulesEditor**, after that the **Rulesdc** tag can be seen.
8. Drag and drop the **rulesdc** tag into the **JSPX** file. You can also add the **rulesDC** tag manually in your jspx file like this:

```
<?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: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="rulesEditor" id="d1">
            <af:form id="f1">
                <af:panelGridLayout id="pgl1" inlineStyle="margin:15px;"
styleClass="AFStretchWidth"
                                    partialTriggers="cb1 cb3 cb4">
                    <af:gridRow id="gr2">
                        <af:gridCell marginStart="5px" marginEnd="5px"
width="100%" halign="stretch" id="gc1">
                            <af:panelGroupLayout id="pgl2" layout="horizontal">
                                <af:commandButton text="Save Dict"
action="#{someBean.saveDictionary}" id="cb1"/>
                                <af:spacer width="10" height="10" id="s2"/>
                                <af:commandButton text="Validate" id="cb3"
action="#{someBean.validate}"
                                                  partialSubmit="true"/>
                                <af:spacer width="10" height="10" id="s8"/>
                                <af:commandButton text="Toggle Mode" id="cb4"
action="#{someBean.toggleMode}"
                                                  partialSubmit="true"/>
                            </af:panelGroupLayout>
                        </af:gridCell>
                    </af:gridRow>
                    <af:gridRow height="100%" id="gr1">
                        <af:gridCell marginStart="5px" marginEnd="5px"
width="100%" halign="stretch" valign="stretch"
                                     id="gc2">
                            <rdc:rulesdc rulesetModel="#{someBean.ruleSetModel}"
id="r1"

ifThenPreferences="#{someBean.ifThenPreferences}"

dtPreferences="#{someBean.dtPreferences}" viewOnly="#{someBean.viewOnly}"
                                         disableVerbalRules="false"></rdc:rulesdc>
                        </af:gridCell>
                    </af:gridRow>
                </af:panelGridLayout>
            </af:form>
        </af:document>
    </f:view>
</jsp:root>
```

How to Refer to the Oracle Rules Shared Libraries

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

To refer to the oracle.rules and the oracle.soa.rules_dict_dc.webapp shared libraries:

1. In Oracle JDeveloper, from the **Application Resources**, open **Descriptors**, and then **META-INF**. Edit the weblogic-application.xml file and add the following lines (this refers to the oracle.rules shared library.)

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

2. In Oracle JDeveloper,

   a. Select **File** menu, then select **New** and then **Deployment Descriptors**.

   b. Select **Weblogic Deployment Descriptor** and select **weblogic.xml** from the list.

   c. Select **version 12.1.2** and click **Finish**.

   d. In weblogic.xml **overview** mode, select **Libraries** from the left and add `oracle.soa.rules_dict_dc.webapp` as the library name. Example weblogicogic.xml file:

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<weblogic-web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.oracle.com/weblogic/weblogic-web-app
http://xmlns.oracle.com/weblogic/weblogic-web-app/1.5/weblogic-web-app.xsd"
xmlns="http://xmlns.oracle.com/weblogic/weblogic-web-app">
  <library-ref>
    <library-name>oracle.soa.rules_dict_dc.webapp</library-name>
  </library-ref>
</weblogic-web-app>
```

   e. Click **Save All**.

Note:

Note that oracle.rules and oracle.soa.rules_dict_dc.webapp shared libraries must be deployed to the embedded WLS server.

---

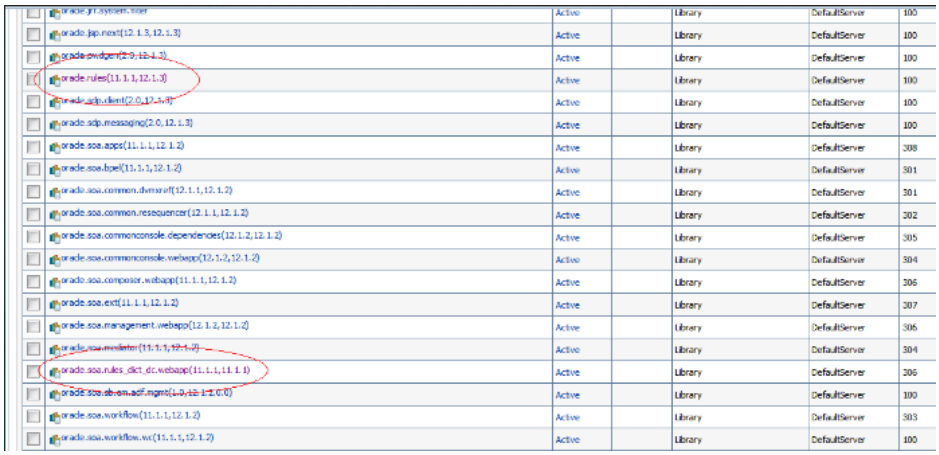
3. All the shared libraries must be deployed using the weblogic console of your embedded WLS:

   a. Launch WLS console (`http://`*`host`*`:`*`port`*`/console/login/LoginForm.jsp`) and log in.

   b. Click **Deployments**.

   Check if **oracle.rules** and **oracle.soa.rules_dict_dc.webapp** shared libraries are deployed as shown in Figure 26-11.

Figure 26-11 Deployments



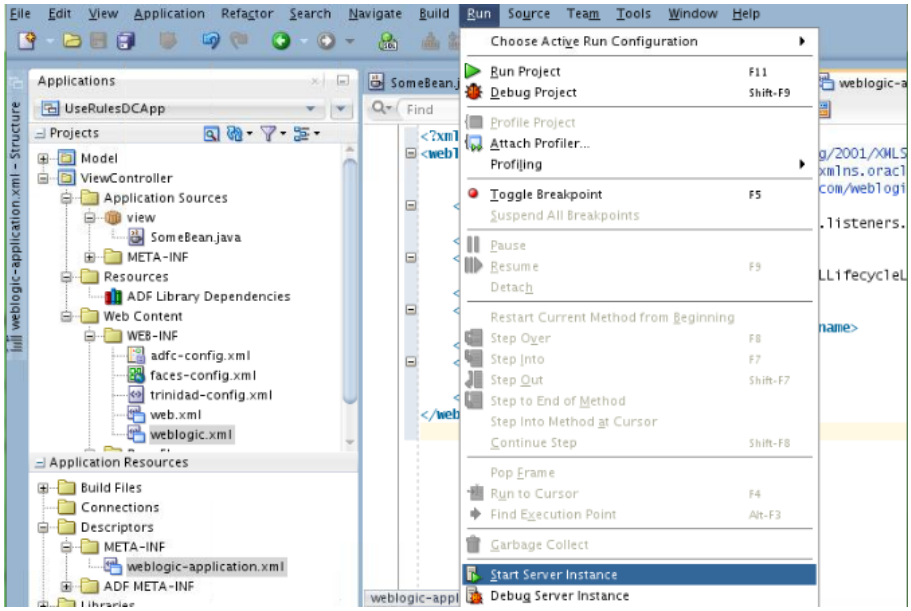
4. Deploy the shared libraries manually if they are not deployed.

   To start the **WLS** embedded server:

   a. Open JDeveloper.

   b. Select **Run** and then select **Start Server Instance** as shown in Figure 26-12

Figure 26-12 Start Embedded WLS



Skip this step if the shared libraries are already deployed.

Note:

WLS embedded server on JDeveloper must be running so that the shared libraries can be deployed.

---

5. To deploy the **oracle.rules** shared library to WLS:

a. Launch WLS console (`http://host:port/console/login/LoginForm.jsp`) and log in.

b. Select **Deployments** and click **Install**.

c. Select **<SOA_INSTALL>/soa/soa/modules/oracle.rules_11.1.1/rules.jar**.

d. Click **Next** and then click **Finish**.

6. To deploy the **oracle.soa.rules_dict_dc.webapp** shared library to WLS:

a. In WLS console, select **Deployments**, click **Install**.

b. Select **<SOA_INSTALL>/soa/soa/modules/oracle.soa.rules_dict_dc.webapp_11.1.1/oracle.soa.rules_dict_dc.webapp.war.**

c. Click **Next** and then click **Finish**.

d. Select **Install this deployment as a library**.

e. Click **Finish**.

f. The **oracle.soa.rules_dict_dc.webapp** gets added to the list of deployments as shown in Figure 26-11

How to Run the Sample Application

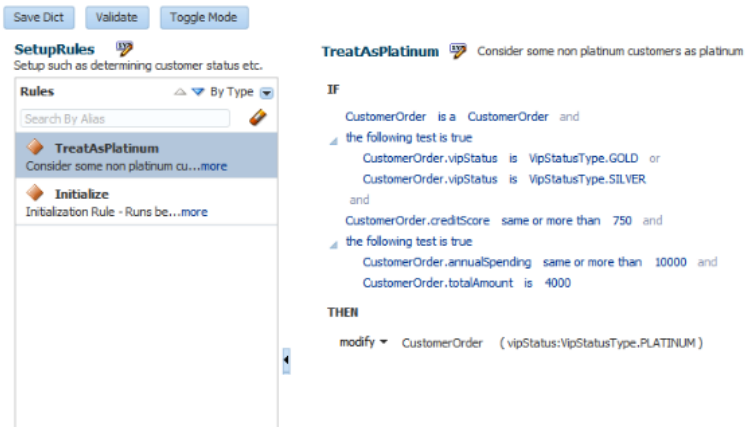
The last task is running the sample application.

To run the Sample Application:

1. To run the sample application, from JDeveloper, right click **rulesEditor.jspx** file.
2. Select **Run**.

   This should start the sample application on a browser., as shown in Figure 26-13.

Figure 26-13 Rules Editor Running



How to Deploy a Rules Editor Application to a Standalone WLS

When you are ready to deploy your application EAR file to the standalone Oracle WebLogic Server, perform the following:

1. Check if the shared libraries are deployed using the weblogic console of your stand-alone WLS.
   a. Launch WLS console. (`http://host:port/console/login/LoginForm.jsp`) and log in.
   b. Click **Deployments**. Check if **oracle.rules** and **oracle.soa.rules_dict_dc.webapp** shared libraries are deployed as showed in Figure 26-11.
2. If the shared libraries are not deployed, then refer to the previous steps to deploy the shared libraries manually.
3. In a project that uses the **Rules Editor Component**:
   a. Include **Oracle Rules Dictionary Component** in your **Libraries and Classpath**.

   This does not deploy these libraries by default, so the jars are not included in your project war file.
4. In a project that is deploying (i.e where you create the ear file):
   a. Add this to your weblogic-application.xml:

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

   b. Add this to weblogic.xml in your project's war file:

```
<library-ref>
    <library-name>oracle.soa.rules_dict_dc.webapp</library-name>
</library-ref>
```

5. Deploy your ear file in WLS.

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*.

What You May Need to Know About the Custom Permissions for the Rules Editor Component

For role-based authorization, Rules DC implements custom JAAS permissions (extending the `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 the **Add Rule** button; if 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 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 permission is not granted, then the rules are view-only.
- `oracle.rules.adf.permission.AddDTPermission`: Displays the **Add Decision Table** button; if 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 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 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:

```
<policy-store>
    <applications>
      <application>
        <name>UseRuleDictDCWtPerm</name>
        <app-roles>
          <app-role>
            <name>Admin</name>
            <class>oracle.security.jps.service.policystore.ApplicationRole</class>
            <display-name>Admin</display-name>
            <members>
              <member>
                <name>admin</name>
                <class>oracle.security.jps.internal.core.principals.JpsXmlUserImpl</
class>
              </member>
            </members>
          </app-role>
          <app-role>
            <name>BusinessUser</name>
            <class>oracle.security.jps.service.policystore.ApplicationRole</class>
            <display-name>BusinessUser</display-name>
            <members>
              <member>
                <name>buser</name>
                <class>oracle.security.jps.internal.core.principals.JpsXmlUserImpl</
class>
              </member>
            </members>
          </app-role>
        </app-roles>
        <jazn-policy>
          <grant>
            <grantee>
              <principals>
                <principal>
```

```
                    <class>oracle.security.jps.service.policystore.ApplicationRole</
class>
                    <name>Admin</name>
                  </principal>
                </principals>
              </grantee>
              <permissions>
                <permission>
                  <class>oracle.rules.adf.permission.RulesEditorPermission</class>
                  <name>RulesEditorPermission</name>
                  <actions>access</actions>
                </permission>
              </permissions>
            </grant>
          </jazn-policy>
        </application>
      </applications>
    </policy-store>
```

If you do not want to use the individual permissions, such as `AddRulePermission` or `DeleteRulePermission`, you can set the `RulesEditorPermission` in the `jazn-data.xml` file to set global permissions.

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 26-1 lists the supported attributes.

Table 26-1 Supported Attributes of the Rules Editor Component

Name	Type	Required	Default Value	Supports EL?	Description
`rulesetModel`	`oracle.bpel.rulesdc.model.interfaces.RuleSetInterface`	yes		Only EL	Wrapper around the Rules SDK ruleset object. The user can use the RuleSetModel object supplied as part of the Rules Editor Component.
`ruleModel`	`java.lang.String`	no	`oracle.bpel.rulesdc.model.impl.RuleModel`	yes	Used to customize the default RuleModel. User can extend the RuleModel class to override certain methods. Deprecated. Use 'ifThenPreferences' attribute and override getRuleModel().

Table 26-1 (Cont.) Supported Attributes of the Rules Editor Component

Name	Type	Required	Default Value	Supports EL?	Description
simpleTestModel	java.lang.String	no	oracle.bpel.rulesdc.model.impl.SimpleTestModel	yes	Used to customize the default SimpleTestModel. User can extend the SimpleTestModel class to override certain methods. Deprecated. Use 'ifThenPreferences' attribute and override getSimpleTestModel().
viewOnly	java.lang.Boolean	no	true	yes	In the "viewOnly" mode user can view the existing rules in the ruleset. If "false", in the "edit" mode, the user is allowed to add new rules and edit existing rules.
genericPattern	java.lang. Boolean	no	true	yes	Deprecated and not used.
genericAction	java.lang.Boolean	no	true	yes	Deprecated and not used.
locale	java.util.Locale	no	Locale.getDefault()	yes	Used for Localization.
timezone	java.util.TimeZone	no	TimeZone.getDefault()	yes	Used for Localization
displayRuleSetEffDate	java.lang.Boolean	no	true	yes	Deprecated and not used.
discloseRules	java.lang.Boolean	no	false	yes	Deprecated and not used.
displayRuleSetName	java.lang.Boolean	no	false	yes	Deprecated and not used.
disableRuleSetName	java.lang.Boolean	no	false	yes	Deprecated and not used.
dtColumnPageSize	java.lang.Integer	no	5	yes	Deprecated and not used.

Table 26-1 (Cont.) Supported Attributes of the Rules Editor Component

Name	Type	Required	Default Value	Supports EL?	Description
dtHeight	java.lang.Integer	no	16	yes	Deprecated and not used.
dateStyle	java.lang.String	no	gets it from the locale	yes	If specified, the date style is used in all inputDate components. Example: "yyyy.MM.dd"
timeStyle	java.lang.String	no	gets it from the locale	yes	If specified, the time style is used in all inutDate components.Example: "HH:mm:ss".
showValidationPanel	java.lang.Boolean	no	true	yes	Displays the validation panel by default. User can choose to hide this by setting this to false.
showDTButtons	java.lang.Boolean	no	true	yes	Deprecated and not used.
rulesPageSize	java.lang.Integer	no	5	yes	Deprecated and not used.
decimalSeparator	java.lang.Character	no	Based on Locale	yes	Used to specify the decimal separators. This is used in Number Formatting. If specified, overrides the decimal separator based on locale.
groupingSeparator	java.lang.Character	no	Based on Locale	yes	Used to specify the grouping separators. This is used in Number Formatting. If specified, overrides the grouping separator based on locale.
disableVerbalRules	java.lang.Boolean	no	true	yes	Disables verbalization UI if 'true'.
vldnPanelCollapsed	java.lang.Boolean	no	false	yes	Used to specify if validation panel should be collapsed by default.

Table 26-1 (Cont.) Supported Attributes of the Rules Editor Component

Name	Type	Required	Default Value	Supports EL?	Description
vldnTabTitle	java.lang.String	no	-	yes	Used to specify the validation panel title.
genericDTAddActionMenu	java.lang.Boolean	yes	true	yes	If 'true', the generic add action menu is displayed in the decision table tool bar. If 'false' consumer must specify the add action menu using 'dtAddActionMenuDDC' attribute. Deprecated. Use 'dtPreferences' attribute and override isGenericDTAddActionMenu().
genericDTEditAction	java.lang.Boolean	no	true	yes	If 'true', generic action UI is displayed in the action editor browser that shows up when an action row is edited in the decision table. If 'false' consumer must specify the edit action UI using the 'dtEditActionDDC' attribute. Deprecated. Use 'dtPreferences' attribute and override isGenericDTEditAction().

Table 26-1 (Cont.) Supported Attributes of the Rules Editor Component

Name	Type	Required	Default Value	Supports EL?	Description
genericDTActionParam	java.lang.Boolean	no	true	yes	If 'true', generic UI is displayed in the action parameter cell of the decision table. If 'false', consumer must specify the action parameter cell UI using the 'dtActionParamCellDDC' attribute. Deprecated. Use 'dtPreferences' attribute and override isGenericDTActionParam().
dtAddActionMenuDDC	jjava.lang.String	no	-	yes	Used only when 'genericDTAddActionMenu' is true. Consumer must pass the DDC (i.e the dynamic declarative component) including the context path that specifies the add menu items in the decision table toolbar. Example "/userulesdc/decisiontable/dtAddActionMenu.jsff". Deprecated. Use 'dtPreferences' attribute and override getDtAddActionMenuDDC()..

Table 26-1 (Cont.) Supported Attributes of the Rules Editor Component

Name	Type	Required	Default Value	Supports EL?	Description
dtEditActionDDC	java.lang.String	no	-	yes	Used only when 'genericDTEditAction' is true. Consumer must pass the DDC (i.e the dynamic declarative component) including the context path that specifies the action UI to be displayed in the action editor browser that shows up when an action row is edited in the decision table. Example "/userulesdc/decisiontable/actionEditor.jsff". Deprecated. Use 'dtPreferences' attribute and override getDtEditActionDDC().
dtActionParamCellDDC	java.lang.String	no	-	yes	Used only when 'genericDTActionParam' is true. Consumer must pass the DDC (i.e the dynamic declarative component) including the context path that specifies the UI to be displayed in the action parameter cell of the decision table. Example "/userulesdc/decisiontable/actionParamCell.jsff". Deprecated. Use 'dtPreferences' attribute and override getDtActionParamCellDDC().

Table 26-1 (Cont.) Supported Attributes of the Rules Editor Component

Name	Type	Required	Default Value	Supports EL?	Description
dtActionNameCustomizer	oracle.bpel.rulesdc.model.interfaces.ActionNameCustomizer	no	-	yes	Used to specify the action name and action parameter name in the decision table header. Deprecated. Use 'dtPreferences' attribute and override getDtActionNameCustomizer().
dtPreferences	oracle.bpel.rulesdc.model.decisiontable.interfaces.DecisionTablePrefs	no	oracle.bpel.rulesdc.model.decisiontable.impl.DecisionTablePrefsImpl	yes	Used to specify decision table preferences. Consumers can extend the default implementation i.e (oracle.bpel.rulesdc.model.decisiontable.impl.DecisionTablePrefsImpl) and override only the required preferences. s.
ifThenPreferences	oracle.bpel.rulesdc.model.interfaces.IfThenPreferences	no	oracle.bpel.rulesdc.model.impl.IfThenPreferencesImpl	yes	Used to specify if validation panel should be collapsed by default.
resourceManager	joracle.bpel.ruless hareddc.model.interfaces.ResourceManagerInterface	no	-	yes	Used to specify the resource manager for translations UI. Refer to the section on 'translations'.
verbalRuleGotoDSLListener	oracle.bpel.rulesshareddc.model.interfaces.VerbalRuleGotoLinkListener	no	-	yes	Listener object triggered when 'Goto phrase' link is clicked from the verbal rule.

Introduction to 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.

Using the Oracle Business Rules Dictionary Component

Rules Dictionary Editor Component is an ADF Declarative Component that allows editing of Business Rules meta-data artifacts such as Rulesets, Value Sets, Globals, Decision Functions and so on using the Rules SDK2 API.

Rules Dictionary Editor Component must not be confused with the Rules Editor Component which is mainly used to edit Rules inside a specified Ruleset. The Rules Dictionary Component is a composite component that allows editing of Globals, Value sets, Rulesets and so on. The Rules Dictionary Editor Task Flow uses the Rules Dictionary Editor Component.

The Rules Dictionary Editor Component provides the following features:

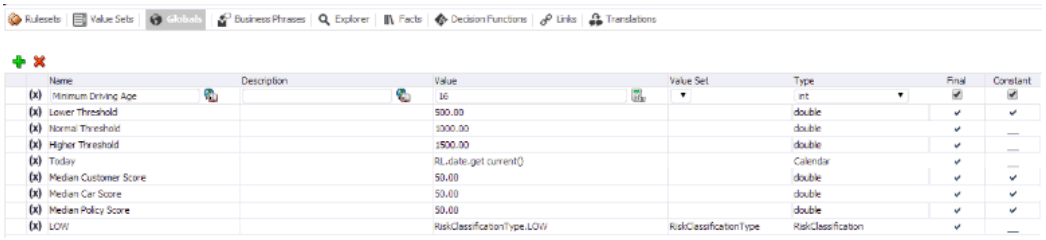
- CRUD (create/read/update/delete) operations on rulesets and general rules, verbal rules and decision tables within a ruleset.
- CRUD (create/read/update/delete) operations on Business Phrases (used in verbalization).
- CRUD (create/read/update/delete) operations on Value sets.
- CRUD (create/read/update/delete) operations on Globals/Variables.
- CRUD (create/read/update/delete) operations on Decision Functions.
- CRUD (create/read/update/delete) operations on RL and XML Facts and viewing for all other Fact types.
- View linked dictionaries.
- Support for user-defined translations.
- Cut/copy/paste of all dictionary components.
- Compare and merge different versions of the dictionary (diff/merge support).
- Export decision tables to Excel.

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 the Rules Dictionary Editor task flow, see Introduction to the Oracle Business Rules Dictionary Editor Task Flow.

The Rules Dictionary Editor component enables you to:

- Edit globals or variables that have the `final` attribute set to `true` by using the Globals Editor, as shown in Figure 26-14.

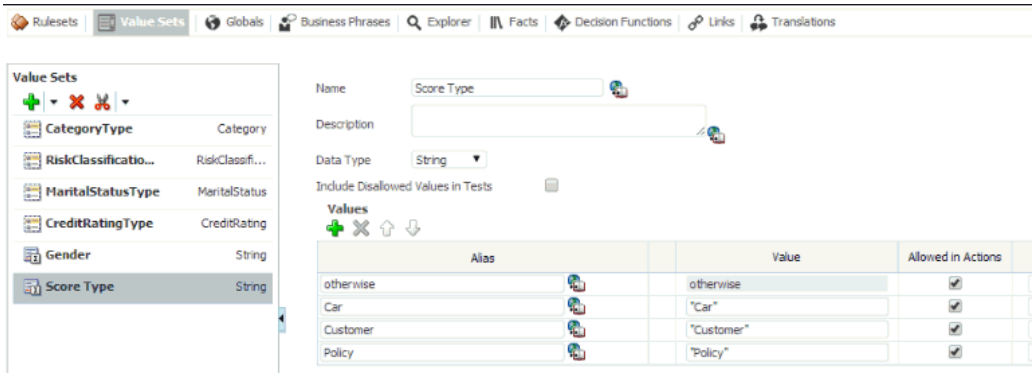
Figure 26-14 Globals Editor



The Globals Editor enables you to create, delete, edit the name, description, value, change value set, change type and make global final. It supports validation of globals.

- Edits value sets by using the Value Sets Editor as shown in Figure 26-15.

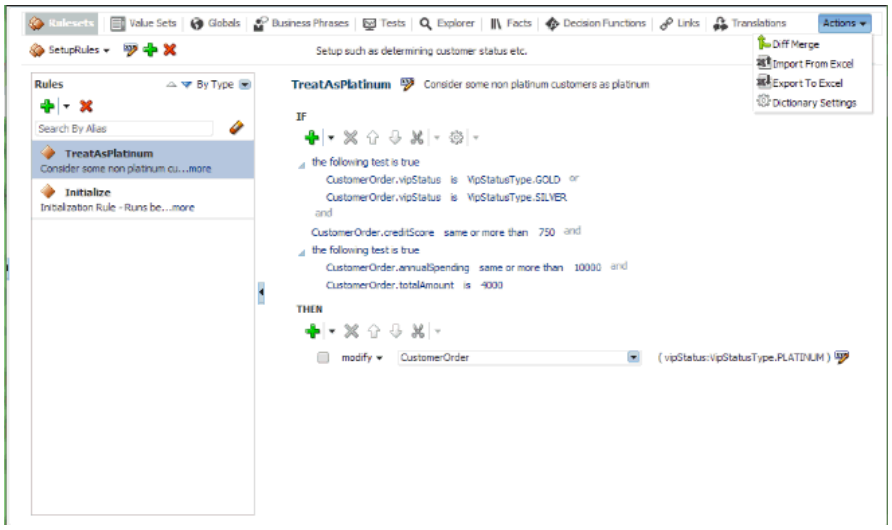
Figure 26-15 Value Sets Editor



The Value Sets Editor enables you to perform CRUD (create, read, update, and delete) operations on value sets and ranges inside a value set. It also supports validation of value sets.

- Edit Rulesets, as shown in Figure 26-16.

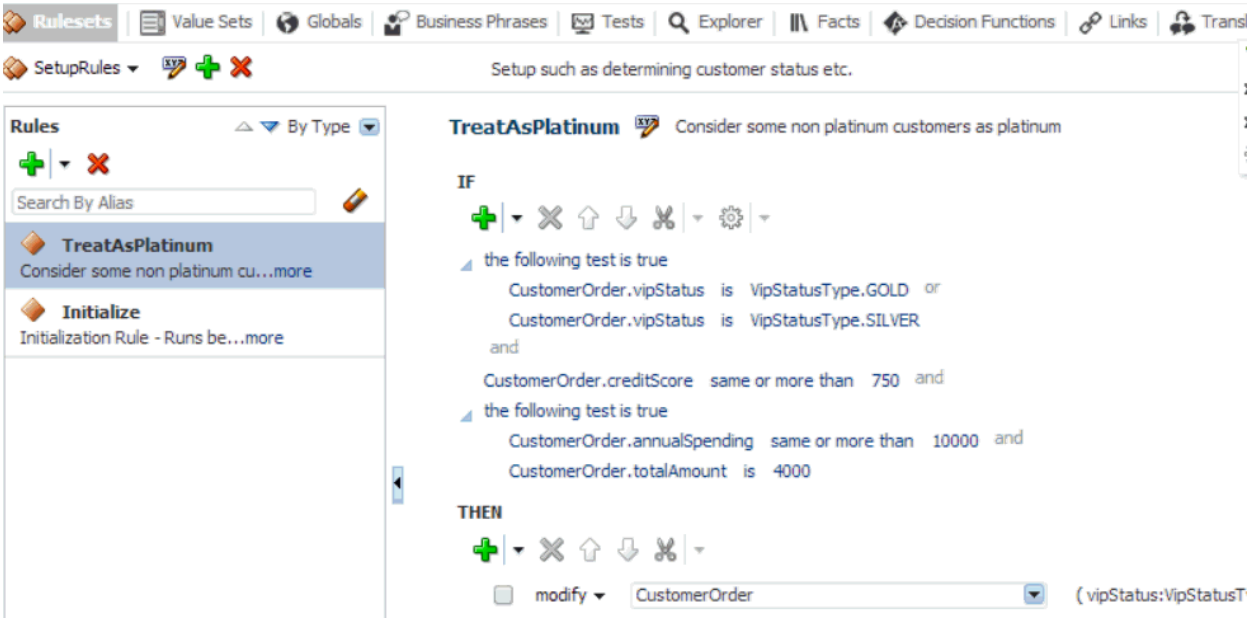
Figure 26-16 Edit Rulesets



The Rules Dictionary Editor enables you to edit only rules inside a selected ruleset. It does not allow creation or deletion of rulesets.

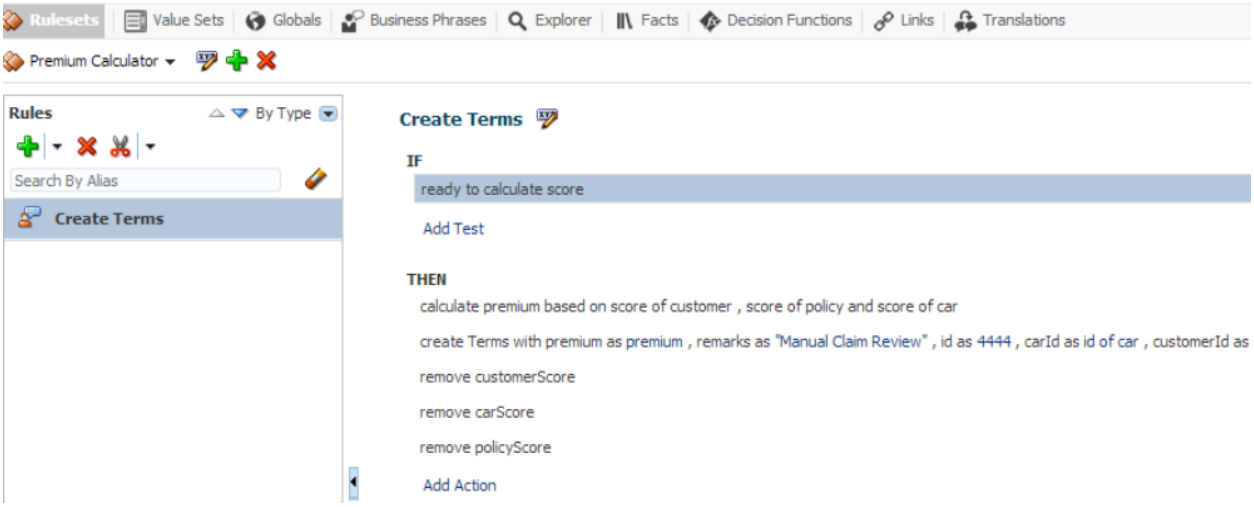
- Edit General Rules, as shown in Figure 26-17.

Figure 26-17 General Rule



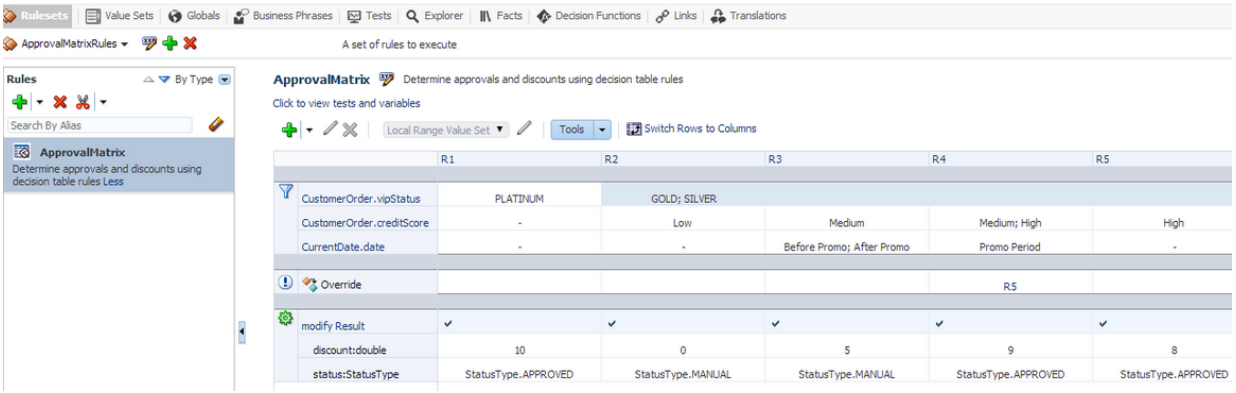
- Edit Verbal Rules, as shown in Figure 26-18.

Figure 26-18 Verbal Rule



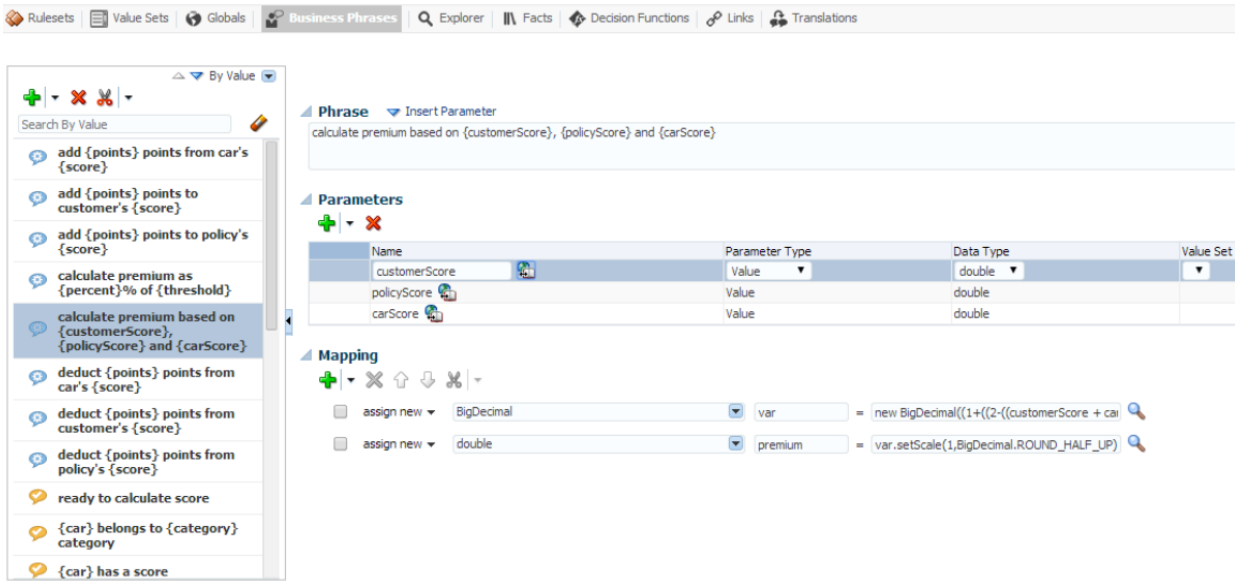
- Edit Decision Tables, as shown in Figure 26-19.

Figure 26-19 Decision Table



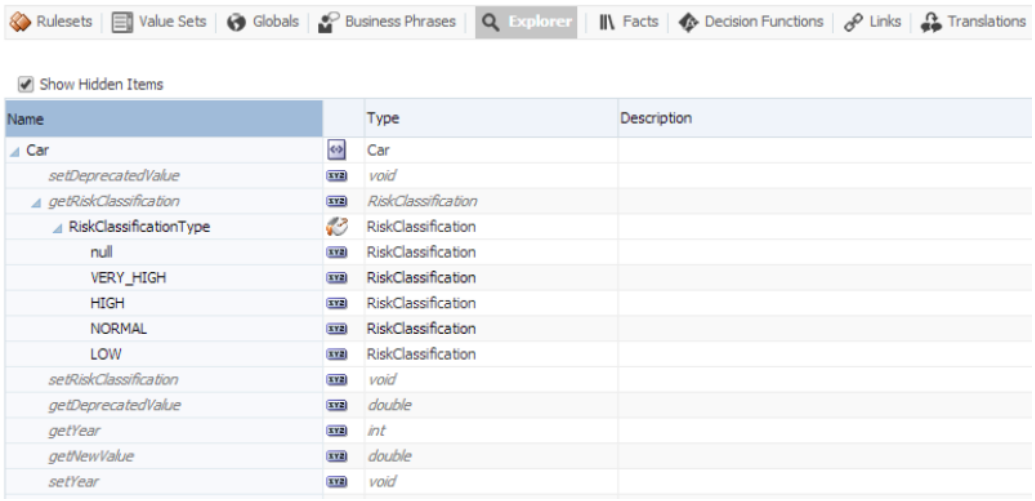
- Edit Business Phrases, as shown in Figure 26-20.

Figure 26-20 Business Phrases Tab



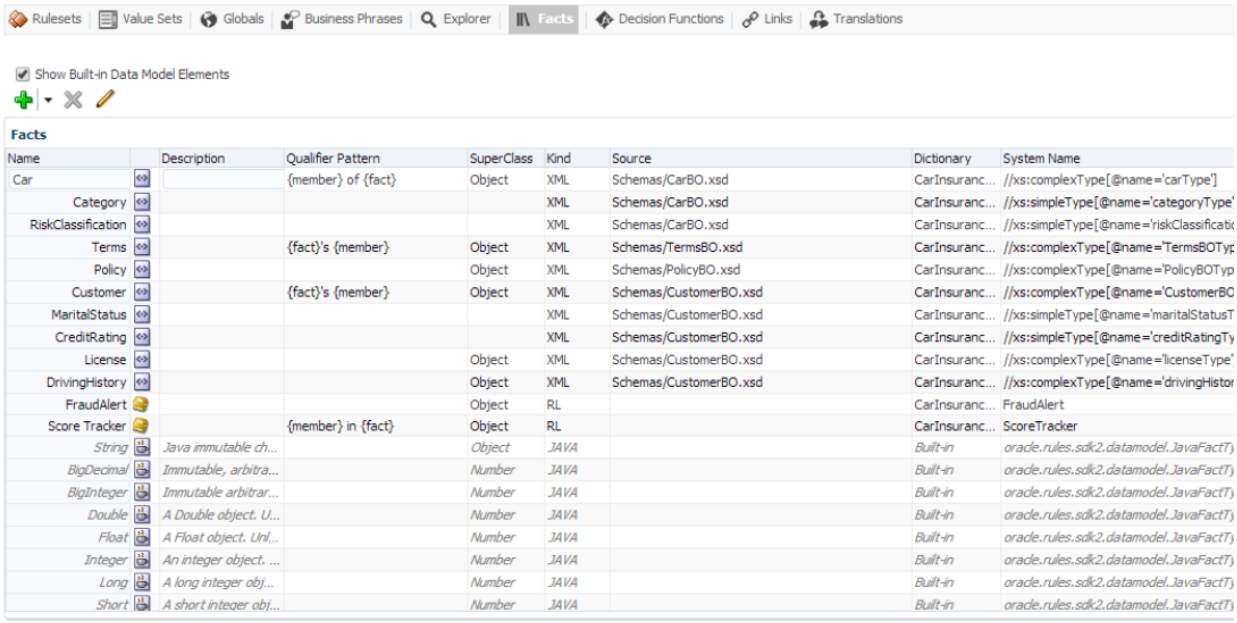
- View Explorer, as shown Figure 26-21.

Figure 26-21 Explorer Tab



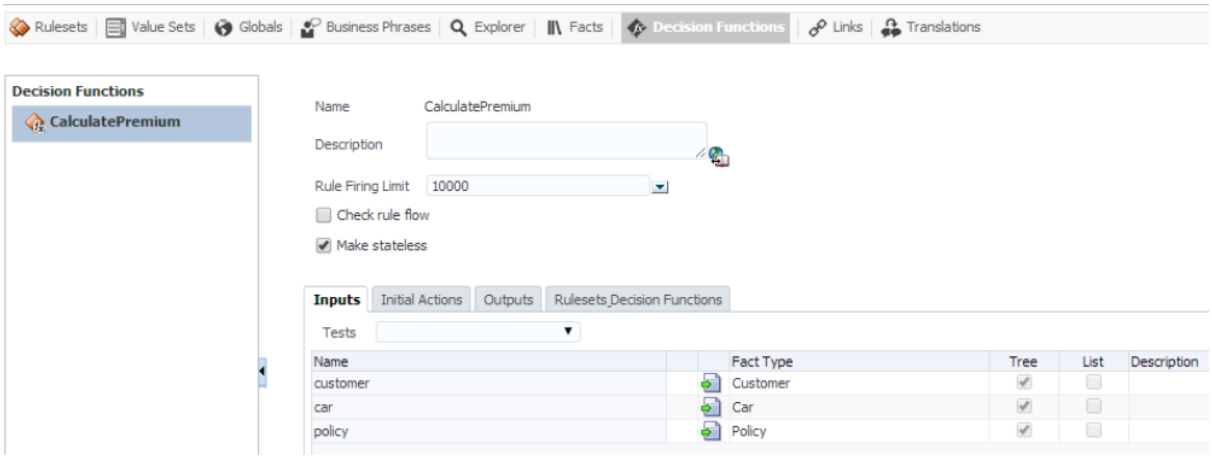
- Edit Facts, as shown in Figure 26-22.

Figure 26-22 Facts Tab



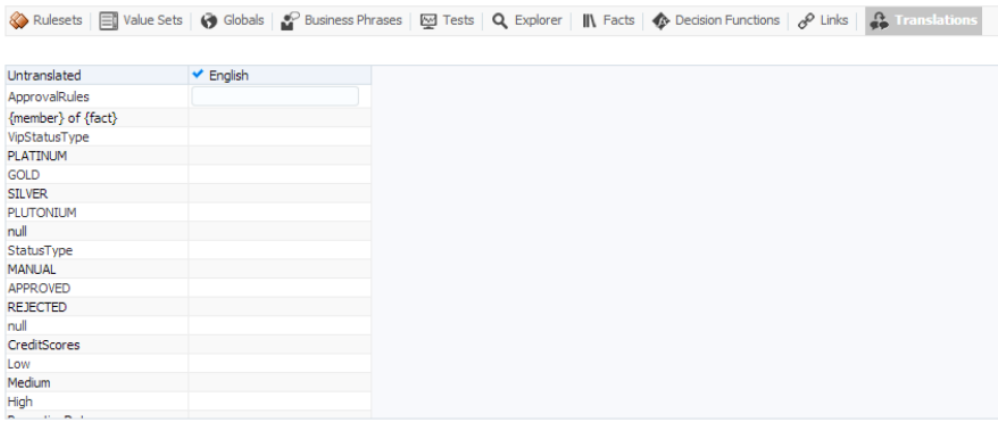
- Edit Decision Functions, as shown in Figure 26-23.

Figure 26-23 Decision Functions Tab



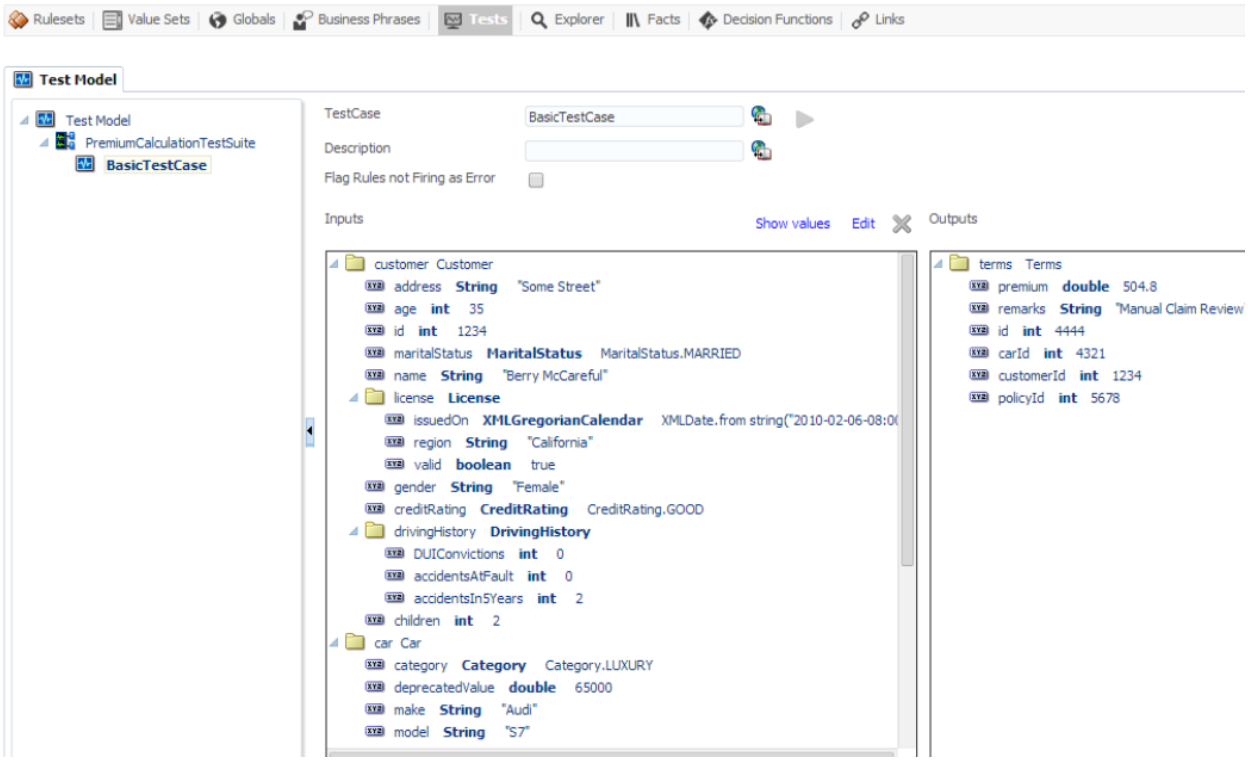
- Edit Translations, as shown in Figure 26-24.

Figure 26-24 Translations Tab



- Create and run tests to validate rules, as shown in Figure 26-25.

Figure 26-25 Tests Tab



For more information about these features and tabs, see *Designing Business Rules with Oracle Business Process Management*.

How to Create and Run a Sample Application by Using the Rules Dictionary Editor Component

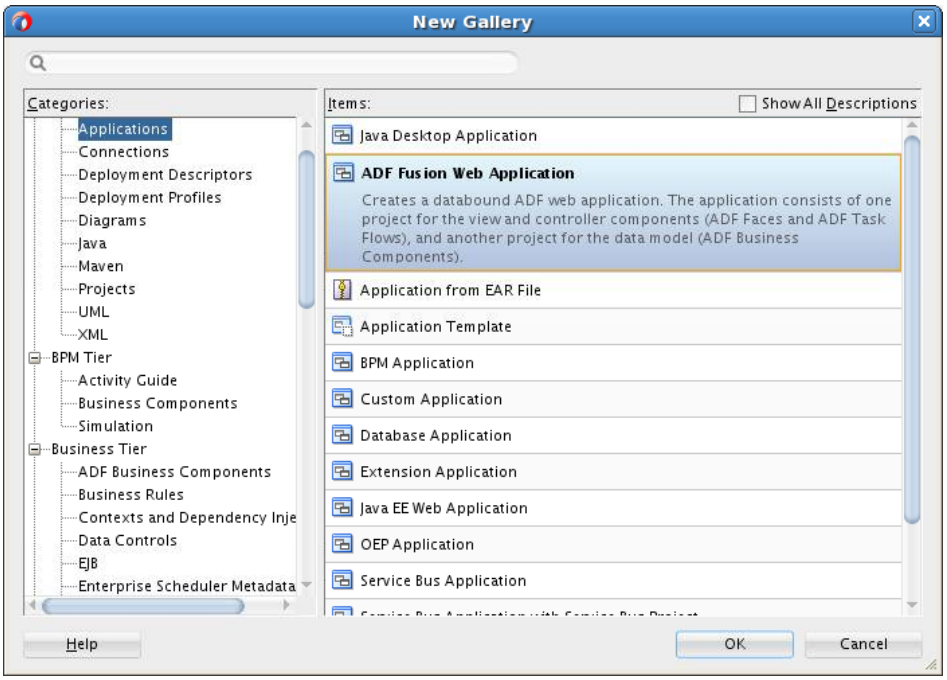
This section lists the steps for creating and running a sample application by using the Rules Dictionary Editor Component.

The prerequisite for using the Rules Dictionary Editor Component to create ADF-based web applications is having JDeveloper with SOA installation.The first task is to create a sample application.

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

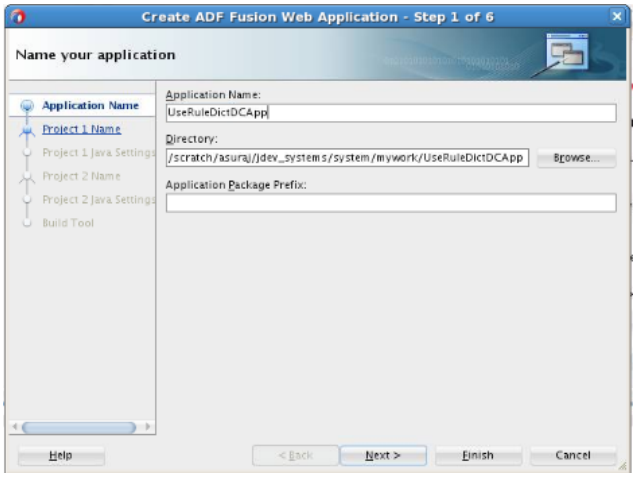
1. Open JDeveloper, from the **File** Menu, select **New** and then select **ADF Fusion Web Application** to create a new application as shown in Figure 26-26.

Figure 26-26 Create Fusion Web Application.



2. Enter a name for the application in the **Application Name** field, for example, `UseRuleDictDCApp` and click **Next** as shown in Figure 26-27.

Figure 26-27 Creating a Generic Application



3. Use the default for everything else.

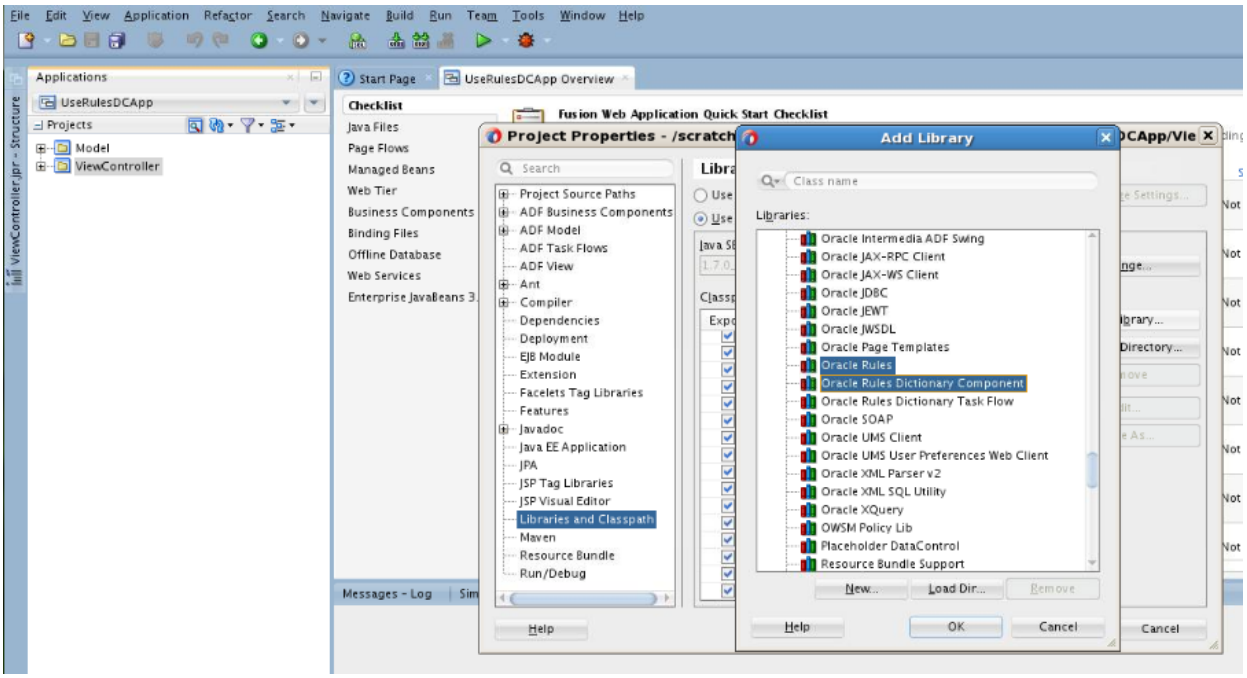
4. Click **Finish**.

5. Right click **ViewController project** and select **Project Properties**. Select **Libraries and Classpath** from the menu on the left.

   a. Click **Add Library**.

   b. Select **Oracle Rules** and **Oracle Rules Dictionary Component** from the Extension List and click **OK**. This adds the Rules SDK and the Rules ADF component tag libraries to the project as shown in Figure 26-28.

Figure 26-28 Adding a Library



Note:

If the 'Oracle Rules' and 'Oracle Rules Dictionary Component' do not show up in the 'Extension' List, open a SOA/BPM project within jDeveloper to load the required libraries.

c. Click **OK** once more to come out of Project Properties.

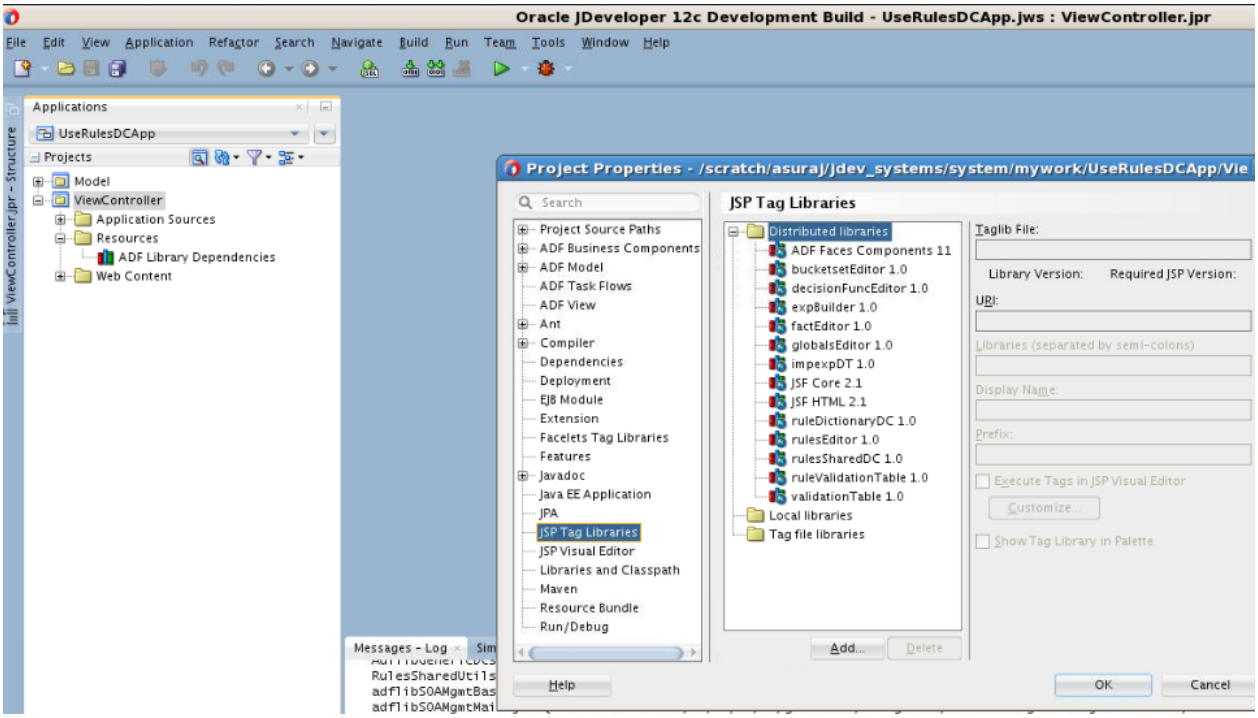
6. Click **Save All** to save the project.

7. Check to make sure all the required tag libraries are added.

   a. Right click **ViewController project** and select **Project Properties**.

   b. Select **JSP Tag Libraries** from the menu on the left and check if all the tag libraries are added as shown in Figure 26-29.

Figure 26-29 JSP Tag Libraries



How to Create the RuleDictionaryModel Object

The Rules Dictionary Editor component requires a `oracle.bpel.ruledictionarydc.model.impl.RuleDictionaryModel` object to create the `RuleDictionaryModel` object.

To create the RuleDictionaryModel object:

1. To create a Java Class e.g. SomeBean.java in your project, from the **File** menu, select **New** and then select **Java Class**.

2. In SomeBean.java provide a method that returns the RuleDictionaryModel object. You must specify the location/path of the rules file. The following is an example of `SomeBean.java`:

```
package view;

import java.io.BufferedReader;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.Serializable;

import java.net.MalformedURLException;
import java.net.URL;

import oracle.bpel.ruledictionarydc.model.impl.RuleDictionaryModel;
import oracle.bpel.ruledictionarydc.model.impl.RulesEditorPreferencesImpl;
import oracle.bpel.ruledictionarydc.model.interfaces.RulesEditorPreferences;
import oracle.bpel.rulesdc.model.decisiontable.impl.DecisionTablePrefsImpl;
import oracle.bpel.rulesdc.model.decisiontable.interfaces.DecisionTablePrefs;
import oracle.bpel.rulesdc.model.impl.IfThenPreferencesImpl;
```

```
import oracle.bpel.rulesdc.model.interfaces.IfThenPreferences;
import oracle.bpel.rulessharedutils.impl.RulesSharedUtils;

import oracle.rules.sdk2.decisionpoint.DecisionPointDictionaryFinder;
import oracle.rules.sdk2.dictionary.DictionaryFinder;
import oracle.rules.sdk2.dictionary.RuleDictionary;
import oracle.rules.sdk2.exception.SDKException;

public class SomeBean {
    private RuleDictionaryModel ruleDictModel;
    private RulesEditorPreferences rulesEditorPrefs;
    private boolean viewOnly = true;

    //on windows
    //private static final String RULES_FILE1 =
 "file:///D:/scratch/asuraj/system_MAIN/rules_
 files/insurancequoteproject/CarInsuranceRules.rules";

    // on linux
    private static final String RULES_FILE1 =
    "file:////scratch/asuraj/backup/rules_files/ApprovalRules.rules";

    public SomeBean() {
        super();
    }

    public RuleDictionaryModel getRuleDictModel() {
        if (ruleDictModel != null)
            return ruleDictModel;

    ruleDictModel = new RuleDictionaryModel(openRulesDict(RULES_FILE1, new
    DecisionPointDictionaryFinder()));
    return ruleDictModel;
    }

    public void saveDictionary() {
        RuleDictionary dict = null;


        if (this.ruleDictModel == null)
            return;
        dict = this.ruleDictModel.getRuleDictionary().getDictionary();


        if (dict == null)
            return;

        if (dict.isModified())
            RulesSharedUtils.updateDictionary(dict);
        if (!dict.isTransactionInProgress())
            saveDictionary(dict, RULES_FILE1);
    }

    public void validate() {
        if (this.ruleDictModel == null)
            return;

        this.ruleDictModel.validate();
    }

    public void toggleMode() {
```

```
        viewOnly = !viewOnly;
    }

    public boolean isViewOnly() {
        return viewOnly;
    }

    public RulesEditorPreferences getRulesEditorPrefs() {
        if (rulesEditorPrefs == null)
            rulesEditorPrefs = new MyRulesEditorPrefs();
        return rulesEditorPrefs;
    }

    //utility methods

    public static RuleDictionary openRulesDict(String fileName, DictionaryFinder
finder) {
        URL url = null;
        try {
            url = new URL(fileName);
        } catch (MalformedURLException e) {
            System.err.println(e);
            return null;
        }
        RuleDictionary dict = null;

        try {
            dict = readFromDisk(url, finder);
        } catch (Exception e) {
            System.err.println(e);
            return null;
        }
        return dict;
    }

    public static RuleDictionary readFromDisk(URL dictURL, DictionaryFinder
 finder) {
        BufferedReader buf = null;
        try {
            buf = new BufferedReader(new
 InputStreamReader(dictURL.openStream(), "UTF-8"));
            return RuleDictionary.readDictionary(buf, finder);
        } catch (SDKException e) {
            System.err.println(e);
        } catch (IOException e) {
            System.err.println(e);
        } finally {
            if (buf != null)
                try {
                    buf.close();
                } catch (IOException e) {
                    System.err.println(e);
                }
        }
        return null;    }    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");

        try {
            writeToDisk(dict, new URL(ruleFileName));
        } catch (MalformedURLException e) {
            System.err.println(e);
            return false;
        } catch (Exception e) {
            System.err.println(e);
            return false;
        }
        return true;
    }

    public static void writeToDisk(RuleDictionary dic, URL dictURL) {
        OutputStreamWriter writer = null;
        try {
            writer = new OutputStreamWriter(new
FileOutputStream(dictURL.getPath()), "UTF-8");
            dic.writeDictionary(writer);
        } catch (IOException e) {
            System.err.println(e);
        } catch (SDKException e) {
            System.err.println(e);
        } finally {
            if (writer != null)
                try {
                    writer.close();
                } catch (IOException e) {
                    System.err.println(e);
                }
        }
    }

    public class MyRulesEditorPrefs extends RulesEditorPreferencesImpl
implements Serializable {

        private DecisionTablePrefs dtPrefs;
        private IfThenPreferences ifThenPrefs;

        @Override
        public DecisionTablePrefs getDecisionTablePreferences() {
            if (dtPrefs == null)
                dtPrefs = new DTPreferences();
            return dtPrefs;
        }

        @Override
        public IfThenPreferences getIfThenPreferences() {
            if (ifThenPrefs == null)
                ifThenPrefs = new MyIfThenPrefs();
            return ifThenPrefs;
        }

        @Override
        public boolean isShowRSButtons() {
            return true;
        }
    }
```

```
   public class MyIfThenPrefs extends IfThenPreferencesImpl implements
Serializable {

       @Override
       public boolean isGenericAction() {
           return true;
       }

       @Override
       public boolean isGenericCondition() {
           return true;
       }
   }    public class DTPreferences extends DecisionTablePrefsImpl implements
Serializable {

       @Override
       public boolean isShowDTButtons() {
           return true;
       }
   }
}
```

3. Point to SomeBean.java in `adfc-config.xml` with Bean Name `someBean` and a **session** scope. Example `adfc-config.xml`.

4. Ensure that **Java Class** under **Items** is selected and click **OK** to display the Create Java Class dialog box.

```
<?xml version="1.0" encoding="UTF-8" ?>
<adfc-config xmlns="http://xmlns.oracle.com/adf/controller" version="1.2">
  <managed-bean id="__1">
    <managed-bean-name>someBean</managed-bean-name>
    <managed-bean-class>view.SomeBean</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
  </managed-bean>
</adfc-config>
```

5. The ADF/JSF framework makes calls to SomeBean.java multiple times to render the UI. For instance, someBean.ruleDictModel is called many times. So it is more efficient to create the ruleDictModel once and cache it and return it each time instead of recreating it.

How to Create .jspx File for the Rules Dictionary Editor Component

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

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

1. Open Oracle JDeveloper.

2. From the **File** Menu, select **New** and then select **JSF/Facelets**.

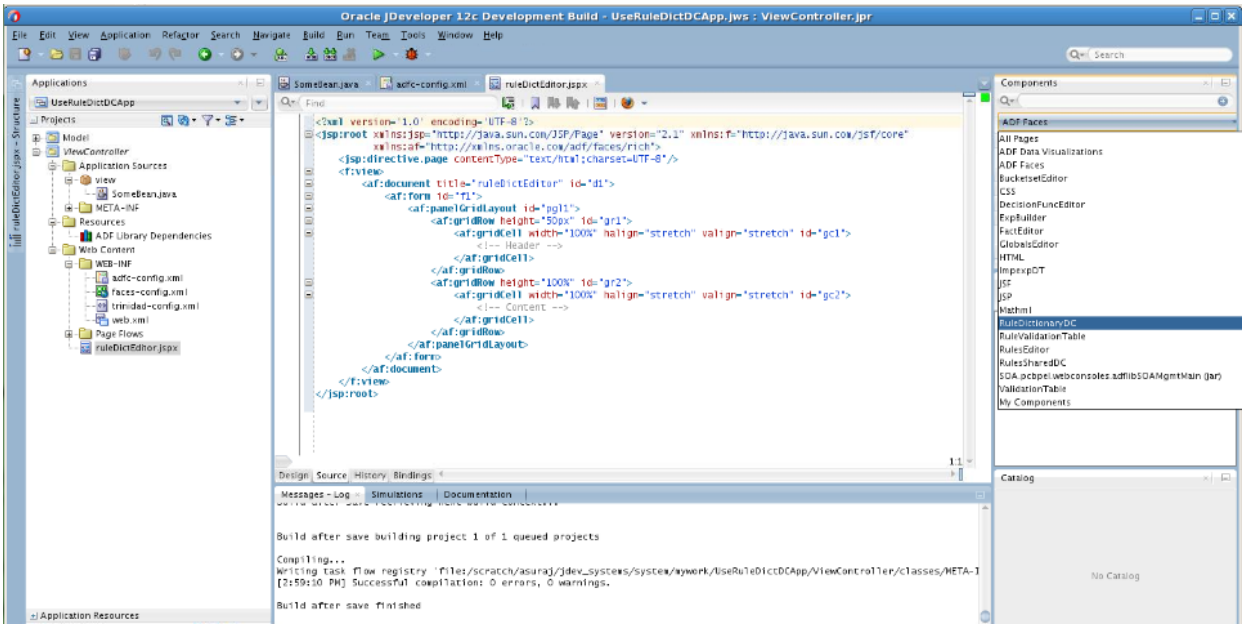
3. Select **JSF Page** and click **OK**.

4. Select Document Type as **JSP XML**.

5. Enter file name as `ruleDictEditor.jspx`. Click **OK**.

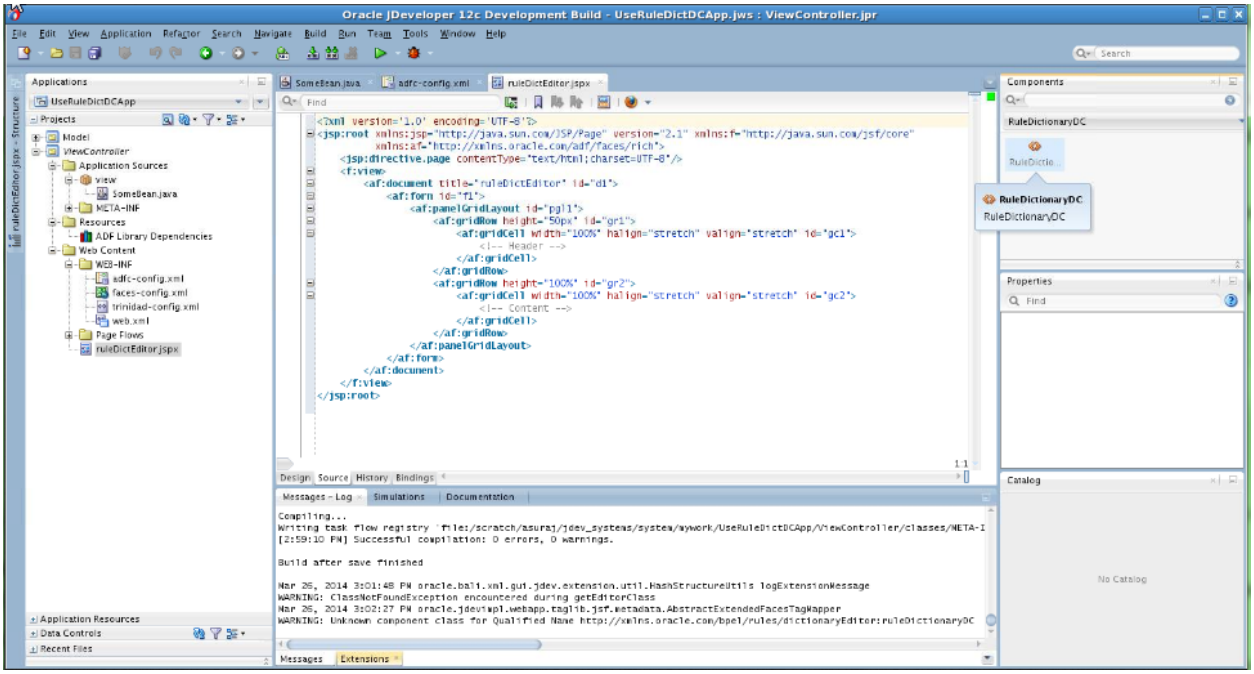
6. The **RuleDictionaryDC** is visible in the Components window in jDeveloper as shown in Figure 26-30.

Figure 26-30 Components Window



7. Select **RuleDictionaryDC**, now you should see the **RuleDictionaryDC** tag. Drag and drop the RuleDictionaryDC tag into the **JSPX** fileFigure 26-31.

Figure 26-31 Rule Dictionary DC Tag



You can also add the 'RuleDictionaryDC' tag manually in your jspx file like this:

```
<?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: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 title="ruleDictEditor" id="d1">
            <af:form id="f1">
                <af:panelGridLayout id="pgl1" inlineStyle="margin:15px;"
styleClass="AFStretchWidth"
                                    partialTriggers="cb2 cb3 cb6">
                    <af:gridRow id="dc_gr1" marginTop="5px" marginBottom="5px">
                        <af:gridCell marginStart="5px" marginEnd="5px"
width="100%" halign="stretch" id="gc1">
                            <af:panelGroupLayout id="pgl3" layout="horizontal">
                                <af:button text="Save Dictionary" id="cb2"
action="#{someBean.saveDictionary}"/>
                                <af:spacer width="10" height="10" id="s1"/>
                                <af:button text="Validate" id="cb3"
action="#{someBean.validate}"/>
                                <af:spacer width="10" height="10" id="s3"/>
                                <af:button text="Toggle Mode" id="cb6"
action="#{someBean.toggleMode}"/>
                            </af:panelGroupLayout>
                        </af:gridCell>
                    </af:gridRow>
                    <af:gridRow height="100%" id="gr2">
                        <af:gridCell width="100%" halign="stretch"
valign="stretch" id="gc2">
                            <!-- Content -->
                            <rddc:ruleDictionaryDC
ruleDictModel="#{someBean.ruleDictModel}" id="rddc1"

rulesEditorPrefs="#{someBean.rulesEditorPrefs}"

viewOnly="#{someBean.viewOnly}" disableVerbalRules="false"/>
                        </af:gridCell>
                    </af:gridRow>
                </af:panelGridLayout>
            </af:form>
        </af:document>
    </f:view>
</jsp:root>
```

How to Refer 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_dict_dc.webapp` shared libraries from the `weblogic-application.xml` file.

To refer to the oracle.rules and the oracle.soa.rules_dict_dc.webapp shared libraries:

1. In JDeveloper from **Application Resources** select **Descriptors** and then **META-INF.** Edit the `weblogic-application.xml` file and add the following lines (this refers to the oracle.rules shared library):

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

2. In JDeveloper select **File**, then **New** and then **Deployment Descriptors.**

   a. Select **Weblogic Deployment Descriptor** and then select **weblogic.xml** from the list.

b. Select **version 12.1.2** and click **Finish**.

c. In weblogic.xml overview mode, select **Libraries** from the left and add library name as `oracle.soa.rules_dict_dc.webapp`. Example weblogic.xml file:

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<weblogic-web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.oracle.com/weblogic/weblogic-web-app
http://xmlns.oracle.com/weblogic/weblogic-web-app/1.5/weblogic-web-app.xsd"
xmlns="http://xmlns.oracle.com/weblogic/weblogic-web-app">
  <library-ref>
    <library-name>oracle.soa.rules_dict_dc.webapp</library-name>
  </library-ref>
</weblogic-web-app>
```

d. Click **Save All**.

Note:

Note that 'oracle.rules' and 'oracle.soa.rules_dict_dc.webapp' shared libraries must be deployed to the embedded WLS server.

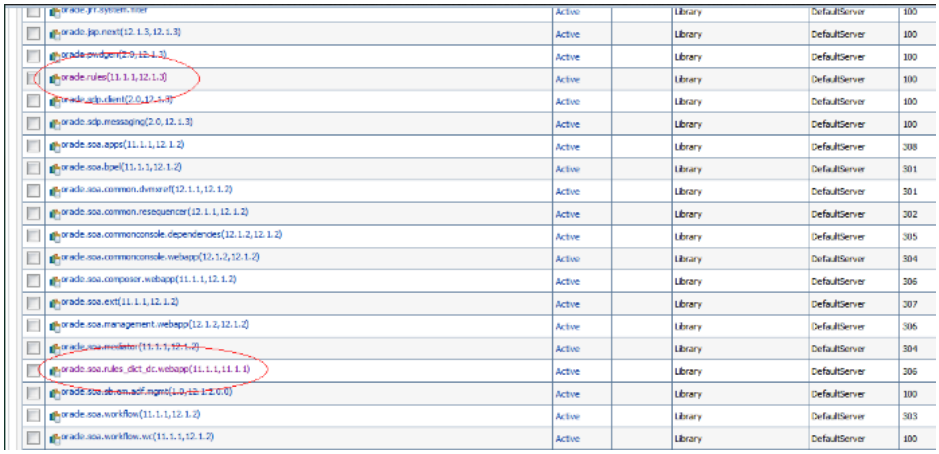
---

3. Check to make sure the shared libraries are deployed using the weblogic console of your embedded WLS.

   a. Launch WLS console (`http://host:port/console/login/LoginForm.jsp`) and log in.

   b. Click **Deployments** and see if **oracle.rules** and **oracle.soa.rules_dict_dc.webapp** shared libraries are deployed as shown in Figure 26-32.

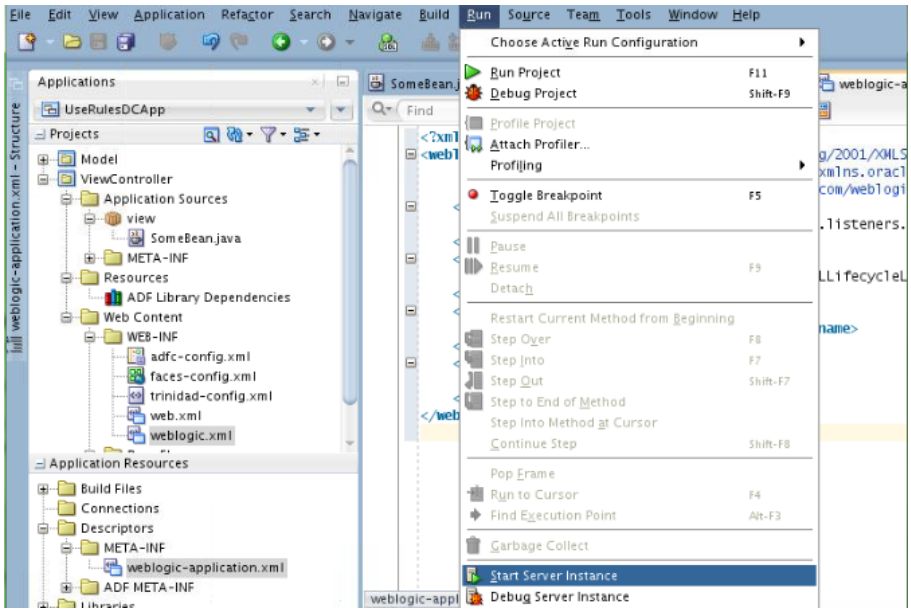
Figure 26-32 Deployments



4. If the shared libraries are not deployed, then follow this process to deploy them manually:

   a. To start the WLS embedded server, in JDeveloper select **Run** and then select **Start Server Instance** as shown in Figure 26-33.

Figure 26-33 Start Server Instance



Skip this if the shared libraries are already deployed.

Note:

WLS embedded server on JDeveloper must be running so that the shared libraries can be deployed.

---

5. To deploy the oracle.rules shared library to WLS:

   a. Launch WLS console (`http://host:port/console/login/LoginForm.jsp`) and log in.

   b. Select **Deployments** and click **Install**.

   c. Select **<SOA_INSTALL>/soa/soa/modules/oracle.rules_11.1.1/rules.jar** and then click **Next** and **Finish**.

6. To deploy the oracle.soa.rules_dict_dc.webapp shared library to WLS:

   a. In WLS console, select **Deployments**, click **Install**.

   b. Select **<SOA_INSTALL>/soa/soa/modules/oracle.soa.rules_dict_dc.webapp_11.1.1/oracle.soa.rules_dict_dc.webapp.war.**

   c. Click **Next** and then click **Finish**.

   d. Select **Install this deployment as a library**.

   e. Click **Finish**

   f. Now you should see **oracle.soa.rules_dict_dc.webapp** added to the list of deployments. as shown in Figure 26-32.

How to Run the Sample Rules Dictionary Editor Application

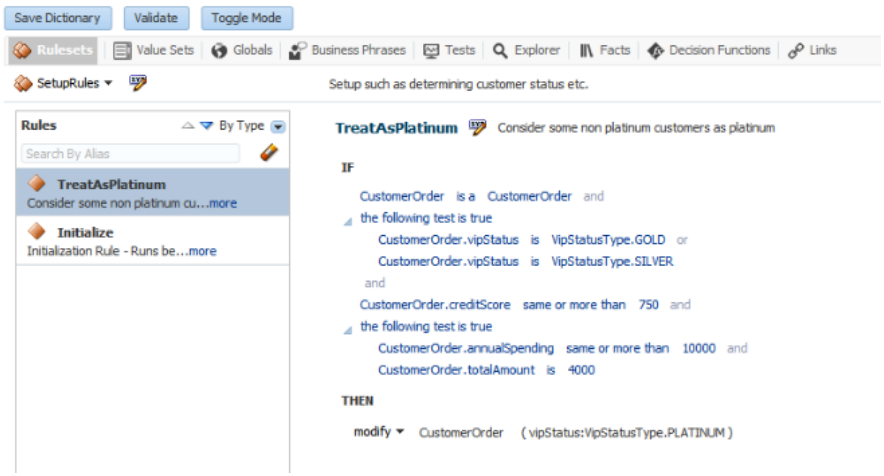
The last task is running the sample application.

To run the sample Rules Dictionary Editor application:

1. To run the sample application, from JDeveloper, right click **ruleDictEditor.jspx** file.
2. Select **Run**.

   This should start the sample application on a browser as shown in Figure 26-34.

Figure 26-34 Rules Dictionary Editor Application



How to Deploy a Rules Dictionary Application to a Standalone Oracle WebLogic Server

When you're ready to deploy your application ear file to the stand-alone WLS, follow these steps to make sure everything runs smoothly.

1. Check to make sure the shared libraries are deployed using the weblogic console of your stand-alone WLS.

   a. Launch WLS console `http://host:port/console/login/LoginForm.jsp` and log in.

   b. Click 'Deployments' and see if 'oracle.rules' and 'oracle.soa.rules_dict_dc.webapp' shared libraries are deployed as shown in Figure 26-11.

2. If the shared libraries are not deployed, then follow the previous process to deploy the shared libraries manually.
3. In your project that uses the Rule Dictionary Editor Component, include the "Oracle Rules Dictionary Component" in your 'Libraries and Classpath'. This does not deploy these libraries by default, so the jars are not included in your project war file.
4. In the project that is finally deploying (i.e where you create the ear file):

   a. Add this to your weblogic-application.xml:

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

b. Add this to weblogic.xml in your project's war file:

```
<library-ref>
    <library-name>oracle.soa.rules_dict_dc.webapp</library-name>
</library-ref>
```

5. Now you can deploy your ear file in WLS and things should work.

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 26-2 lists the supported attributes.

Table 26-2 Supported Rules Dictionary Editor Attributes

Name	Type	Required	Default Value	Supports EL?	Description
`ruleDictModel`	`oracle.bpel.ruledictionarydc.model.interfaces.RuleDictionaryInterface`	yes	-	Only EL	Wrapper around the Rules SDK Dictionary object.The user can use the RuleDictionaryModel object supplied as part of the Rules Dictionary Editor Component jar file (adflibRuleDictionaryDC.jar).
`viewOnly`	`java.lang.Boolean`	no	`true`	yes	In the "viewOnly" mode user can view the existing dictionary data but cannot edit. If "false", i.e. the "edit" mode, the user is allowed to edit the dictionary.
`locale`	`java.util.Locale`	no	`Locale.getDefault()`	yes	Used for Localization.
`timezone`	`java.util.TimeZone`	no	`TimeZone.getDefault()`	yes	Used for Localization

Table 26-2 (Cont.) Supported Rules Dictionary Editor Attributes

Name	Type	Required	Default Value	Supports EL?	Description
ruleModel	java.lang.String	no	oracle.bpel.rulesdc.model.impl.RuleModel	yes	Used to customize the default RuleModel. User can extend the RuleModel class to override certain methods. Deprecated. Use 'rulesEditorPrefs' and override getIfThenPreferences(). getRuleModel().
simpleTestModel	java.lang.String	no	oracle.bpel.rulesdc.model.impl.SimpleTestModel	yes	Used to customize the default SimpleTestModel. User can extend the SimpleTestModel class to override certain methods. Use 'rulesEditorPrefs' and override getIfThenPreferences(). getSimpleTestModel().
selectedTab	java.lang.String	no	-	yes	Switches to the specified tab name (either GLOBALS, FACTS, VALUESETS, LINKS, DESC_FUNCS, DSL_DEFNS, TESTS, TRANSLATIONS or the ruleset name).
selectedRulesetIdx	java.lang.String	no	-	yes	Used to specify the ruleset index to be selected by default. If 'selectedRulesetIdx' is specified, it overrides the 'selectedTab' attribute.
dtColumnPageSize	java.lang.Integer	no	5	yes	Deprecated and not used.
dtHeight	java.lang.Integer	no	16	yes	Deprecated and not used.
dateStyle	java.lang.String	no	gets it from the locale	yes	If specified, the date style is used in all inputDate components. Example: "yyyy.MM.dd".

Table 26-2 (Cont.) Supported Rules Dictionary Editor Attributes

Name	Type	Required	Default Value	Supports EL?	Description
`timeStyle`	`java.lang.String`	no	gets it from the locale	yes	If specified, the time style is used in all inutDate components.Example: "HH:mm:ss".
`showValidationPanel`	`java.lang.Boolean`	no	`true`	yes	Displays the validation panel by default. User can choose to hide this by setting this to false.
`discloseRules`	`java.lang.Boolean`	no	`false`	yes	Deprecated and not used.
`displayRuleSetName`	`java.lang.Boolean`	no	`true`	yes	Deprecated and not used.
`disableRuleSetName`	java.lang.Boolean	no	`false`	yes	Deprecated and not used.
`showDTButtons`	`java.lang.Boolean`	no	`true`	yes	Deprecated and not used
`disableDFName`	`java.lang.Boolean`	no	`false`	yes	Disables the Decision Function Name in the Decision Function editor pop-up if true. Deprecated. Use 'dfEditorPrefs' and override isDisableDFName().
`displayWSName`	`java.lang.Boolean`	no	`true`	yes	Displays the decision service name if 'true' in the Decision Function editor pop-up. Note that the service name makes sense only when 'Invoke as rule service' is checked. Deprecated. Use 'dfEditorPrefs' and override isDisplayWSName()..
`displayWSCheck`	`java.lang.Boolean`	no	`true`	yes	Displays the 'Invoke as rule service' check box in the Decision Function editor pop-up if true. Deprecated. Use 'dfEditorPrefs' and override isDisplayWSCheck().

Table 26-2 (Cont.) Supported Rules Dictionary Editor Attributes

Name	Type	Required	Default Value	Supports EL?	Description
disableInputOps	java.lang.Boolean	no	false	yes	Disables add, edit and delete operations for the Inputs table in the Decision Function editor pop-up. Deprecated. Use 'dfEditorPrefs' and override isDisableInputOps().
disableOutputOps	java.lang.Boolean	no	false	yes	Disables add, edit and delete operations for the Outputs table in the Decision Function editor pop-up. Deprecated. Use 'dfEditorPrefs' and override isDisableOutputOps().
displayAddDF	java.lang.Boolean	no	true	yes	Displays the add decision function button. Deprecated. Use 'dfEditorPrefs' and override isDisableAddDF().
displayDeleteDF	java.lang.Boolean	no	true	yes	Displays the delete decision function button. Deprecated. Use 'dfEditorPrefs' and override isDisableDeleteDF().
rulesPageSize	java.lang.Integer	no	5	yes	Deprecated and not used.
decimalSeparator	java.lang.Character	no	Based on Locale	yes	Used to specify the decimal separators. This is used in Number Formatting. If specified, overrides the decimal separator based on locale.
groupingSeparator	java.lang.Character	no	Based on Locale	yes	Used to specify the grouping separators. This is used in Number Formatting. If specified, overrides the grouping separator based on locale.

Table 26-2 (Cont.) Supported Rules Dictionary Editor Attributes

Name	Type	Required	Default Value	Supports EL?	Description
`vldnPanelCollapsed`	`java.lang.Boolean`	no	`false`	yes	Used to specify if validation panel should be collapsed by default.
`vldnTabTitle`	`java.lang.String`	no	Localized text "Business Rule Validation - Log"	yes	Used to specify the validation panel title.
`resourceManager`	`oracle.bpel.rulessharedc.model.interfaces.ResourceManagerInterface`	no	-	yes	Used to specify the resource manager for translations UI. Refer to the section on 'translations'.
`rulesEditorPrefs`	`oracle.bpel.ruledictionarydc.model.interfaces.RulesEditorPreferences`	no	`oracle.bpel.ruledictionarydc.model.impl.RulesEditorPreferencesImpl`	yes	Used to specify the rules editor preferences. Consumers can extend the default implementation i.e (oracle.bpel.ruledictionarydc.model.impl.RulesEditorPreferencesImpl) and override only the required preferences.
`dfEditorPrefs`	`oracle.bpel.ruledictionarydc.model.interfaces.DFEditorPreferences`	no	`oracle.bpel.ruledictionarydc.model.impl.DFEditorPreferencesImpl`	yes	Used to specify the decision function editor preferences. Consumers can extend the default implementation i.e (oracle.bpel.ruledictionarydc.model.impl.DFEditorPreferencesImpl) and override only the required preferences.
`showRSButtons`	`java.lang.Boolean`	no	`true`	yes	Deprecated and not used.
`dfListener`	`oracle.bpel.decisionfunceditordc.listener.DecisionFuncListener`	no	-	yes	Used for notification of decision function editor updates. Deprecated. Use 'dfEditorPrefs' and override getDfListener().

Table 26-2 (Cont.) Supported Rules Dictionary Editor Attributes

Name	Type	Required	Default Value	Supports EL?	Description
dfActionListener	oracle.bpel.ruledictionarydc.listener.DecisionFuncActionListener	no	-	yes	Used for notification when a decision function is added or deleted. Deprecated. Use 'dfEditorPrefs' and override getDfActionListener()
dfServiceNameCustomize	oracle.bpel.decisionfunceditordc.listener.DecisionFuncServiceNameCustomizer	no	-	yes	Used to customize the decision function service name. Deprecated. Use 'dfEditorPrefs' and override getDfServiceNameCustomizer().
dictVersionInfo	oracle.bpel.ruledictionarydc.model.interfaces.DictVersionInfo	no	-	yes	Used in diff/merge to retrieve the list of dictionary versions for comparison. Deprecated. Use 'dfEditorPrefs' and override getDfServiceNameCustomizer().
testExecutor	oracle.bpel.testeditordc.interfaces.TestExecutor	no	-	yes	Used for executing test suites, test templates and test cases.
disableRulesTesting	java.lang.Boolean	no	false	yes	If true, the rule testing capability is disabled.
disableVerbalRules	java.lang.Boolean	no	true	yes	If true, the verbalization capability is disabled that is the Business Phrases tab is not displayed and CRUD operations on verbal rules are disabled.
boUpdateListener	oracle.bpel.ruledictionarydc.listener.BOUpdateListener	no	-	yes	Used for synchronizing business objects.

Introduction to the Oracle Business Rules Dictionary Editor 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.

Using the Oracle Business Rules Dictionary Task Flow

The Oracle 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 the Rules Dictionary Editor component, see Introduction to the Oracle Business Rules Dictionary Editor Declarative Component.

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 Oracle Rules Dictionary Editor task flow.

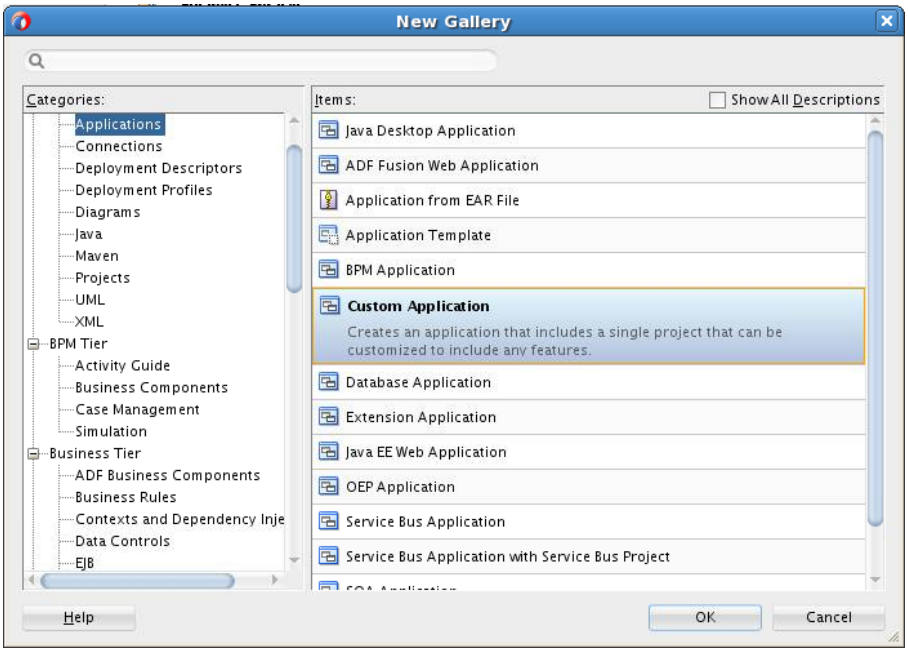
The prerequisites for using the Oracle Rules Dictionary Editor task flow to create ADF-based web applications is having a running installation of Oracle SOA Suite and Oracle JDeveloper on your computer.

The first task is to create a sample application.

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

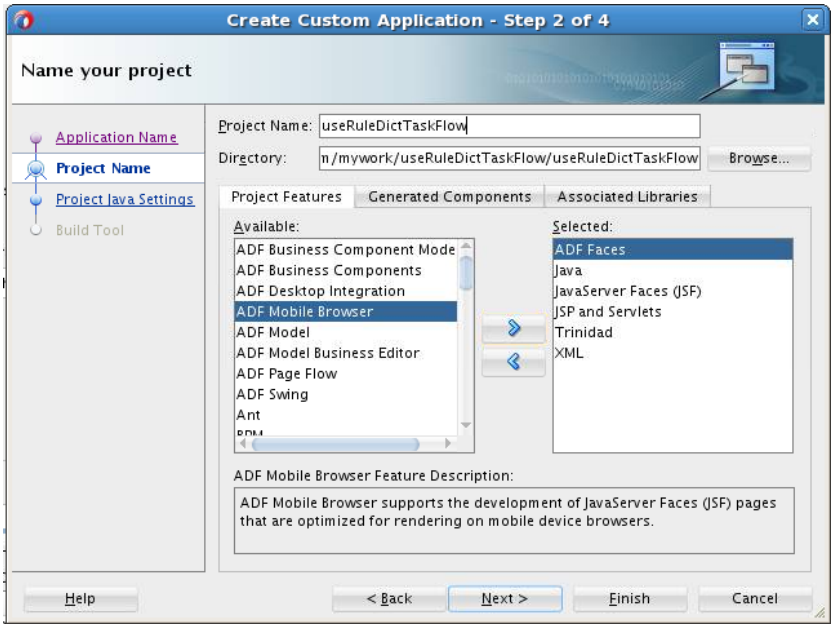
1. Open Oracle JDeveloper.
2. From the **File** menu, select **New** and then **Custom Application** to create an application.
3. Enter a name for the application in the **Application Name** field, for example, `useRuleDictTaskFlowApp`, and click **Next** as shown in Figure 26-35.

Figure 26-35 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 26-36.

Figure 26-36 Creating a Task Flow Project



5. Click **Finish** to create the project.

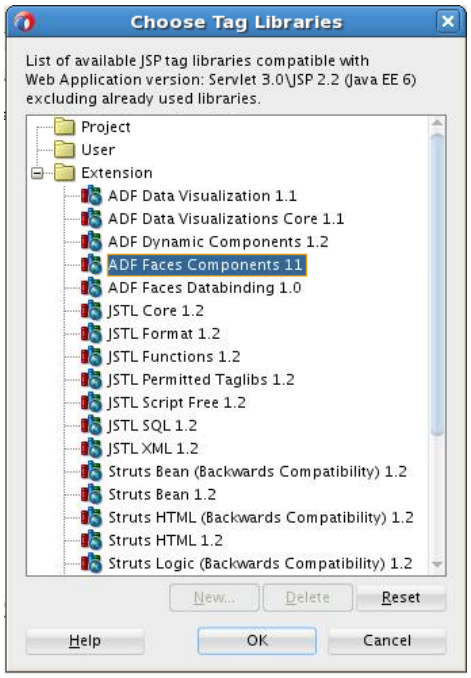
6. Right-click the **useRuleDictTaskFlow** project in the Applications window of Oracle JDeveloper, and select **Project Properties** to display the Project Properties dialog box.

   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 26-37.

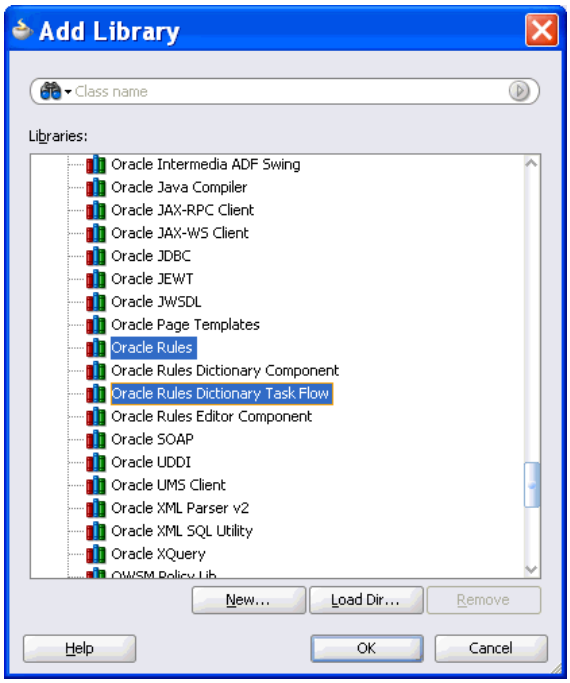
Figure 26-37 Choosing Tab Libraries for the Task Flow Application



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 26-38. This adds the Rules SDK and the Rules Dictionary Task Flow JARs to the project.

Figure 26-38 Adding the Rules SDK and Rules Dictionary Task Flow



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

7. Click **Save All** from the Oracle JDeveloper **File** menu to save the project.

8. 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 The MetadataDetails Interface.

   The steps are:

   a. Open Oracle JDeveloper.

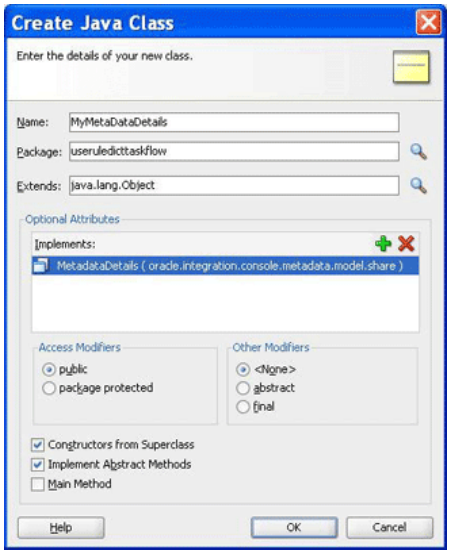
   b. From the **File** menu, select **New** to display the **New Gallery** dialog box.

   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`.

   e. 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 26-39.

Figure 26-39 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 "";
        }
    }
```

9. 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 The NLSPreferences Interface.

   The following sample of `MyNLSPreferences.java` 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;
    }
}
```

10. 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:

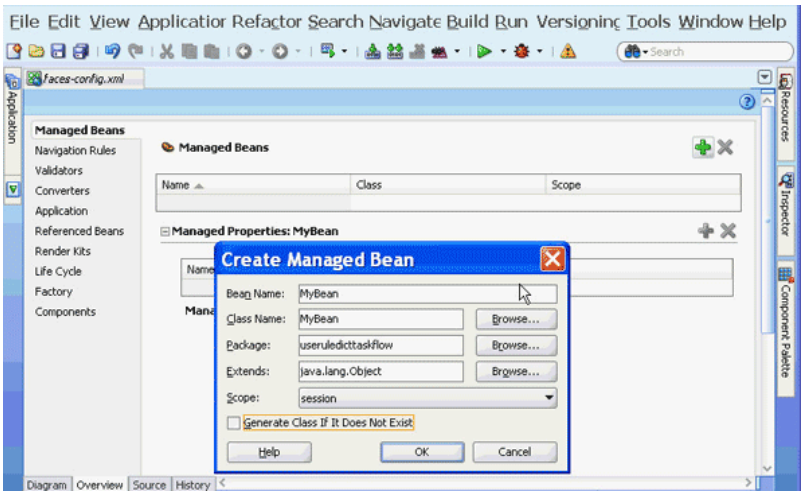
```
package useruledicttaskflow;

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;
    }
}
```

11. Open the `faces-config.xml` file in **Overview** mode and click the **+** button under **Managed Beans** to display the Create Managed Bean dialog box.

12. Point to `MyBean.java` by entering `MyBean` in the **Bean Name** field and selecting `session` from the **Scope** list, as shown in Figure 26-40.

Figure 26-40 Specifying the Bean Name and Scope in the Task Flow Application



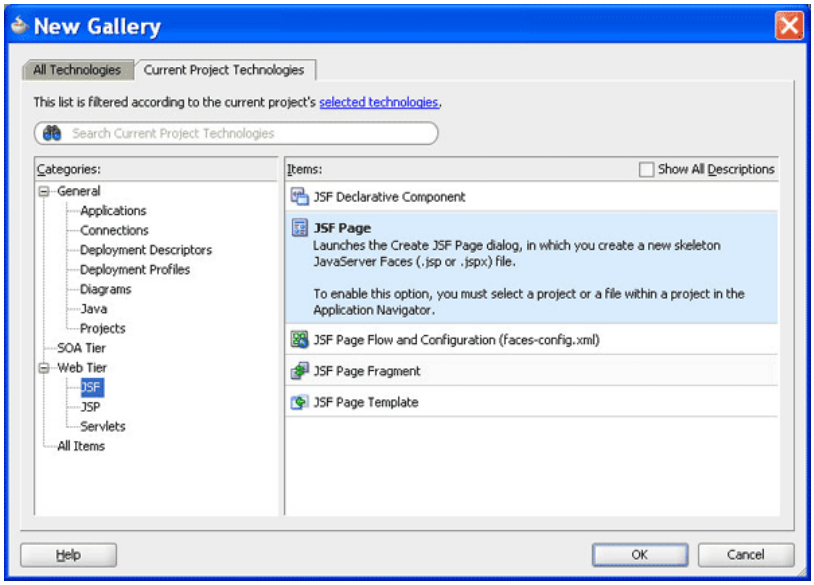
How to Add a Rule Dictionary Editor Task Flow

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

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

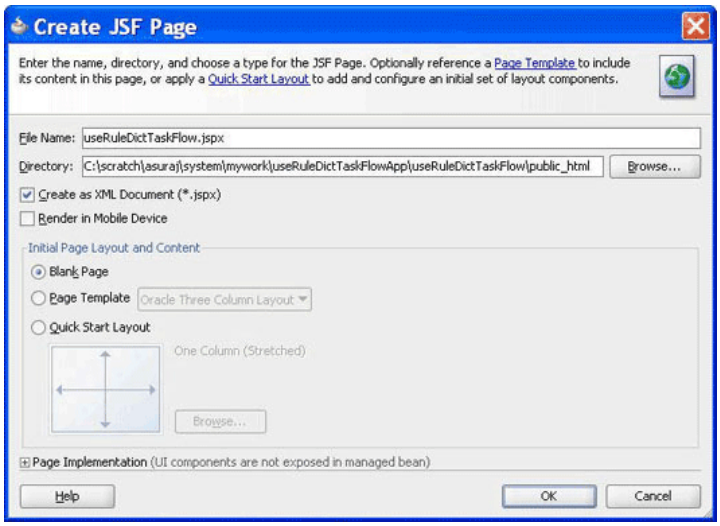
1. Open Oracle JDeveloper.
2. From the **File** menu, select **New** to display the New Gallery dialog box.
3. In the **New Gallery** dialog box, select **JSF** under **Web Tier** from the **Categories** panel.
4. Select **JSF Page** under **Items** and click **OK** to display the **Create JSF Page** dialog box, as shown in Figure 26-41.

Figure 26-41 Creating the JSF Page File to Include the Rules Dictionary Editor Task Flow



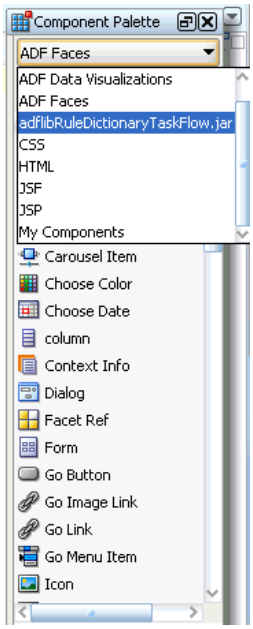
5. In the Create JSF Page dialog box, enter `useRuleDictTaskFlow.jspx` as the file name, as shown in Figure 26-42.

Figure 26-42 Specifying the Name of the JSF Page for the Task Flow



adflibRuleDictionaryTaskFlow.jar is displayed in the Components window of Oracle JDeveloper, as shown in Figure 26-43.

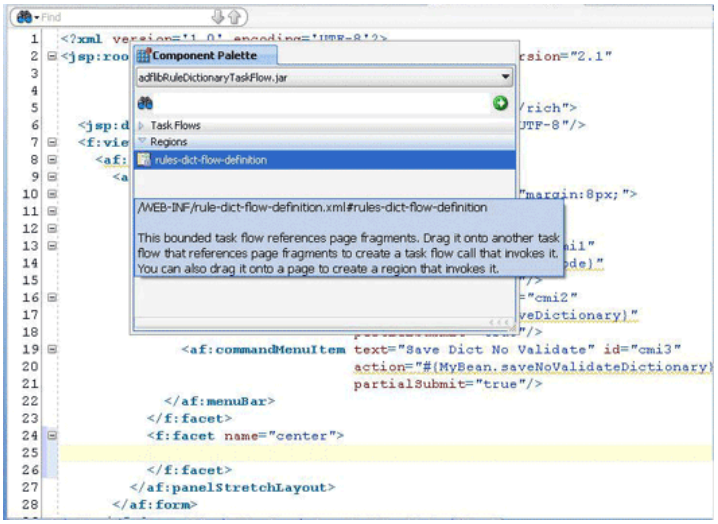
Figure 26-43 Rules Dictionary Task Flow JAR in the Components Window



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** available under **Regions** in the Components window. You can drag and drop the **rule-dict-flow-definition** region into the **.jspx** file as shown in Figure 26-44, and specify all the required parameters.

Figure 26-44 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 Oracle SOA Composer.
- **Save Dict**: Enables saving the dictionary (with or without validation).

How 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
```

How to Refer to oracle.rules and 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 How to Create and Run a Sample Application by Using the Rules Dictionary Editor Component.

How to Run the Sample Task Flow Application

The last task is running the sample application in the embedded Oracle WebLogic Server.

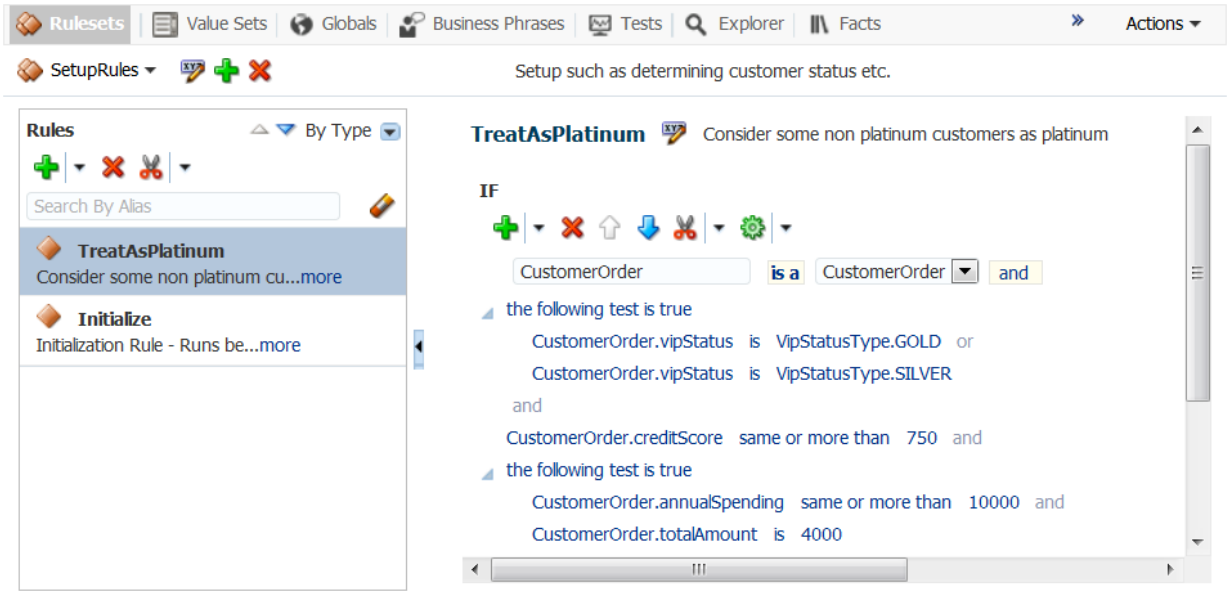
To run the sample task flow application:

1. To run the sample application, from Oracle JDeveloper, right-click the **useRulesDictTaskFlow.jspx** file.

2. Select **Run**.

   This starts the sample application in a web browser, as shown in Figure 26-45.

Figure 26-45 Running the Sample Rules Dictionary Editor Task Flow Application



How to Deploy a Rules Dictionary Editor Task Flow Application to a Standalone Oracle WebLogic Server

When you are ready to deploy your application EAR file to the standalone Oracle WebLogic Server, perform the following:

1. Launch the Oracle WebLogic Server Administration Console (`http://host:port/console/login/LoginForm.jsp`).
2. Ensure that `oracle.rules` is displayed in the deployments list.
3. Ensure that `oracle.soa.rules_dict_dc.webapp` is displayed in the deployments list.
4. 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.
5. 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 Oracle WebLogic Server.

Localizing the ADF-Based Web Application

You can localize an application that is created using the Rules Editor component, Rules Dictionary Editor component, or Rules Dictionary Editor task flow.

To localize your application:

1. Change the `faces-config.xml` in the application using the Rule Dict Editor component. The `faces-config.xml` file should have the following code within the `<application>` tag in order 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 you are interested in.
3. If you want to override the locale provided by the browser and display the UI in some particular locale then pass that locale as an attribute to the component and modify the f:view tag to the following in the application using the component.:

```
<f:view locale="#{someBean.locale}">
```

Note:

The locale passed here should be same as the one passed to the component using 'locale' attribute.

Working with Translations

This feature supports translation of aliases in Business Rules Web UI. You can have the aliases according to the locale. You can also edit the translations of aliases for different locales through translation tab or resource editor pop-up in Business Rules Web UI.

Enabling Translations for Consumer of Reusable Rules UI ADF Task Flow Component

To support translation of aliases, the consumers of reusable Rules UI ADF Task Flow component must provide locale specific resource artifacts as additional parameters while calling Rules UI ADF Task Flow. However, these additional parameters are optional and required only if the consumers want to use the enhanced translation support.

The additional parameters are:

```
property-name: relatedDetails
property-class:
oracle.integration.console.metadata.model.share.IRelatedMetadataDetails

<taskFlow id="rulesdictflowdefinition1"
              taskFlowId="/WEB-INF/rule-dict-flow-definition.xml#rules-dict-flow-
definition"
              activation="deferred" Refresh="default"
              RefreshCondition="${MyBean.refreshReqd}"
              xmlns="http://xmlns.oracle.com/adf/controller/binding">
 <parameter id="relatedDetails"
                    value="#{MyBean.relatedMetadataDetails}"/>
</taskflow>
```

Sample Code to Pass an Implementation of `IRelatedMetadataDetails`

The consumer has to pass an implementation of **oracle.integration.console.metadata.model.share.IRelatedMetadataDetails**

The implementation of `IRelatedMetadataDetails` contains the code for loading the resource bundles from the repository and also for saving the bundles files when user commits any change to rules application.

The consumer should use `dictionaryName + "Translations_" + locale.toString() + ".xml"` convention to build the name of the resource bundle file.

```
public class MyRelatedMetadataDetails implements IRelatedMetadataDetails {

  private static final Locale[] LOCALES = { Locale.US, Locale.FRENCH };

  private static final String RESOURCE_PATH =
    "file:///C:/scratch/sumit/system/rules/";
  private static final String RESOURCE_BASE = "SimpleRule";

  public MyRelatedMetadataDetails() {
    super();
  }

  public String getDocument(IRelatedMetadataPath relatedPath) {
    String resourceSuffix = relatedPath.getValue();
    try {
```

```
    return loadResource(resourceSuffix);
  } catch (IOException e) {
    return "";
  }
}

private static String loadResource(String resourceSuffix) throws IOException {

  FileInputStream fis = null;
  FileChannel fc = null;
  try {
    URL url = new URL(RESOURCE_PATH + RESOURCE_BASE + resourceSuffix);
    fis = new FileInputStream(url.getFile());
    fc = fis.getChannel();
    ByteBuffer bb = ByteBuffer.allocate((int)fc.size());
    fc.read(bb);
    bb.rewind();
    return Charset.defaultCharset().decode(bb).toString();
  } finally {
    if (fis != null) {
      fis.close();
    }
    if (fc != null) {
      fc.close();
    }
  }
}

public void createDocument(IRelatedMetadataPath relatedPath,
                           String document) {
  try {
    storeResource(relatedPath.getValue(), document);
  } catch (IOException e) {
    e.printStackTrace();
  }
}

public void saveDocument(IRelatedMetadataPath path, String document) {
  try {
    storeResource(path.getValue(), document);
  } catch (IOException e) {
    e.printStackTrace();
  }
}

private static void storeResource(String resourceSuffix,
                                  String document) throws IOException {
  FileOutputStream fos = null;
  FileChannel fc = null;
  try {
    URL url = new URL(RESOURCE_PATH + RESOURCE_BASE + resourceSuffix);
    fos = new FileOutputStream(url.getFile());
    fc = fos.getChannel();
    ByteBuffer bb = ByteBuffer.allocateDirect(1024);
    bb.clear();
    bb.put(Charset.defaultCharset().encode(document));
    bb.flip();
    while (bb.hasRemaining()) {
      fc.write(bb);
    }
  } finally {
```

```
      if (fos != null) {
        fos.close();
      }
      if (fc != null) {
        fc.close();
      }
    }
  }

  public IRelatedMetadataPathFinderFactory getFinderFactory() {
    return new RelatedMetadataPathFinderFactory();
  }

  public List<IRelatedMetadataPath> getExisting(IRelatedMetadataPathFinder finder) {

    List<IRelatedMetadataPath> paths = new ArrayList<IRelatedMetadataPath>();
    for (Locale locale : LOCALES) {
      paths.add(RelatedResourceMetadataPath.buildFromLocale(locale));
    }
    return paths;
  }

  public class RelatedMetadataPathFinderFactory implements
IRelated`MetadataPathFinderFactory {

    public IRelatedMetadataPathFinder getResourceFinder() {
      return new RelatedMetadataPathFinder();
    }
  }

  public class RelatedMetadataPathFinder implements IRelatedMetadataPathFinder {

    public String getType() {
      return null;
    }

    public IRelatedMetadataPath
matches(oracle.integration.console.metadata.model.share.MetadataPath srcPath,

oracle.integration.console.metadata.model.share.MetadataPath matchPath) {
      return null;
    }
  }

}
```

Enabling Translations for Consumer of Rules Web UI Application

To support translation of aliases, the consumer of Rules Web UI application must pass an attribute to the Rules Dictionary DC or Rules DC. The attribute is `resourceManager` which accepts an instance of type **oracle.bpel.rulessharedc.model.interface.ResourceManagerInterface.java**. However, this additional parameters are optional and required only if the consumers want to use the enhanced translation support.

```
<rddc:ruleDictionaryDC ruleDictModel="#{SomeBean.ruleDictModel1}"
  id="rddc1"
   resourceManager="#{SomeBean.resourceManager}">
  </rddc:ruleDictionaryDC>
```

Sample Code for Creating an Instance of resourceManager

Implementation of `ResourceManagerInterface` is provided as **oracle.bpel.rulessharedc.model.impl.ResourceManager**. Consumers may create an instance of ResourceManager and pass it to corresponding UI component.

Note:

The consumer has to load all the saved resource bundles from the repository and should construct a java.util.Map (resourceMap) where java.util.Locale of the resource bundle is kept as key and the content of the resource bundle file as value which is of type java.lang.String.

The consumer should use `dictionaryName + "Translations_" + locale.toString() + ".xml"` convention to build the name of the resource bundle file.

The consumer has to save these resource bundles to the repository whenever the user commits any change in the application.

```
public ResourceManagerInterface getResourceManager() {
    if (resourceManager == null) {
      resourceManager =
          new ResourceManager(loadResources(), ruleDictionary);
    }
    return resourceManager;
  }

  private Map<Locale, String> loadResources() {

    Map<Locale, String> resourceMap = new HashMap<Locale, String>();

    for (Locale locale : LOCALES) {
      try {
        URL url =
          new URL(RULES_FILE_PATH + "Translations_" + locale.toString() +
                  ".xml");
        String content =
          new Scanner(new File(url.getFile()), "UTF-8").useDelimiter("\\A").next();
        resourceMap.put(locale, content);
      } catch (IOException e) {
        resourceMap.put(locale, "");
        LOG.severe("Failed to load resource:" + e.getMessage());
      }
    }
    if (!resourceMap.keySet().contains(getLocale())) {
      resourceMap.put(getLocale(), "");
    }
    return resourceMap;
  }

  private void storeResources(Map<Locale, String> resourceMap) {
    for (Locale locale : resourceMap.keySet()) {
      try {
        URL url =
          new URL(RULES_FILE_PATH + "Translations_" + locale.toString() +
                  ".xml");
        BufferedWriter out = new BufferedWriter(new FileWriter(url.getFile()));
```

```
        out.write(resourceMap.get(locale));
        out.close();
      } catch (IOException e) {
        LOG.severe("Failed to store resource:" + e.getMessage());
      }
    }
  }
```

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:

27

Getting Started with Human Workflow

This chapter describes for developers the 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:

Warning:

You must not modify SOA Human Task database tables directly. Oracle does not guarantee backward compatibility for the column names and data in these tables.

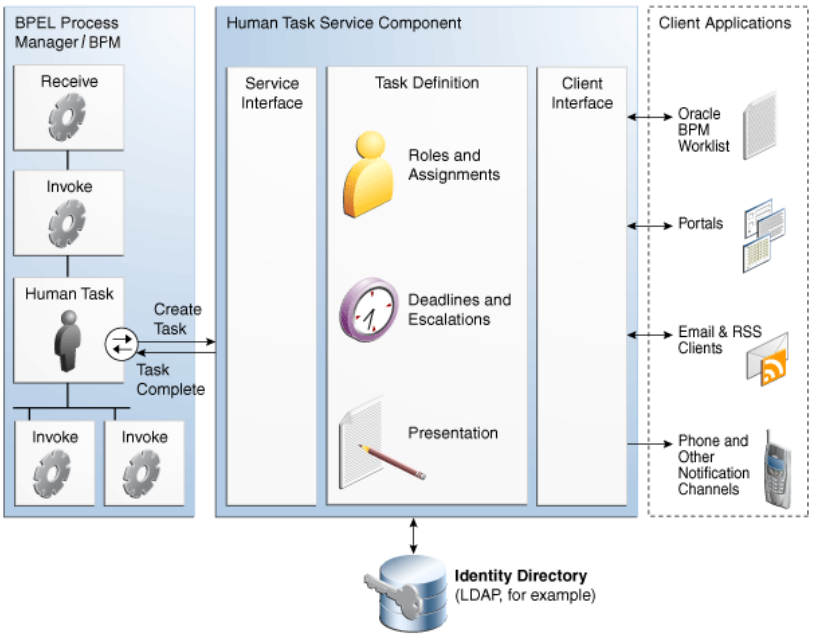
Introduction to Human Workflow

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 27-1 provides an overview of human workflow.

Figure 27-1 Human Workflow



In Figure 27-1, 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 to determine people's roles and privileges.

  You can configure the identity store to use the embedded WebLogic LDAP, Oracle Virtual Directory, third-party LDAPs and Active Directory RDBMS. For more information, see *Securing Applications with Oracle Platform Security Services*.
- 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.

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.

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.

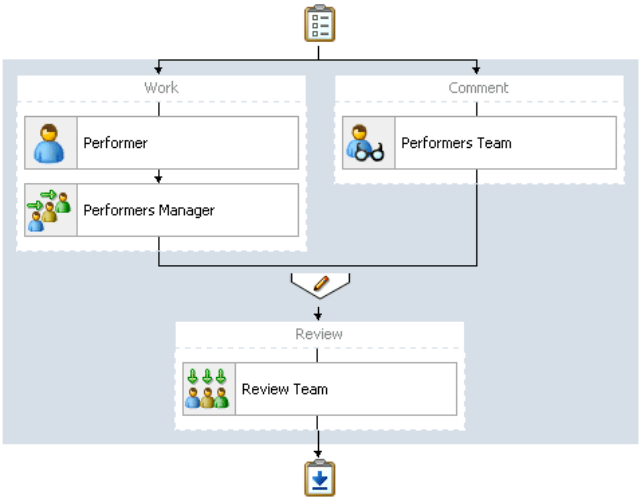
Note:

Human workflow design-time tasks are performed in a graphical editor known as the Human Task Editor. The tutorial in Human Workflow Tutorial describes how to use this editor.

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 27-2). Human workflow provides declarative, pattern-based support for such scenarios.

Figure 27-2 Participants in a Task



Participant

A participant is a user or set of users in the assignment and routing policy definition. In Figure 27-2, each block with an icon representing people is a participant.

Participant Type

In simple cases, a participant maps to a user, group, or role. However, as discussed in 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 Assigning Task Participants.

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 Task Forms.

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 Allow All Participants to Invite Other Participants or Edit New Participants.

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 27-2). 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 Stopping Routing of a Task to Further Participants.

Static, Dynamic, and Rule-Based Task Assignment

There are different methods for assigning users, groups, and application roles to tasks.

- Static Task Assignment
- Dynamic Task Assignment

Static Task Assignment

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`).

Dynamic Task Assignment

You can assign users, groups, and application roles dynamically in the following ways:

- By using a task-assignment pattern. This pattern enables you to do the following:
  - Simply enable participants to claim the task manually. This is the default behavior. No task-assignment pattern is applied.
  - If the participant type is either Single or FYI, then apply a task-assignment pattern to select a single assignee of a requested type from *all* potential assignees in the participant.

    For example, suppose that the potential assignees comprise the user `jcooper`, the group `LoanAgent`, and the application role `Developers`. Suppose further that the requested type is `user`. Applying this task-assignment pattern selects a single user from the user `jcooper`, and from all members of the group `LoanAgent`, and from all users with the application role `Developers`.

  - If the particulates type is Parallel or Serial, then apply a task-assignment pattern to select a single assignee of a requested type from *each* of the potential assignees in the participant.

    For example, suppose that the potential assignees comprise the user `jcooper`, the group `LoanAgent`, and the application role `Developers`. Suppose further that the requested type is `user`. Applying this task-assignment pattern selects the user `jcooper`, and one user from the group `LoanAgent`, and one user with the application role `Developers`.

- By using XPath expressions. These expressions enable you to dynamically determine assignment to users not included in the participant type. Here you create a list of potential assignees, one of whom must then claim the task.

  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 a single user, group, or application role or 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`.

You can use both options simultaneously—for example, you can use an XPath expression to dynamically select a group, and then apply a task-assignment pattern to dynamically select a user from that group.

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. You can also apply the task-assignment pattern to a participant list created using business rules.

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 How to Specify a Task Owner to specify an owner in the Human Task Editor or Specifying a Task Owner to specify an owner in the **Advanced** tab of the Human Task dialog box.

- 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 Specifying the Task Initiator and Task Priority.

- Reviewer

  This participant can review the status of the task and add comments and attachments. You can grant the reviewer role to a participant at runtime using the process instance attributes reviewer and reviewerType. The reviewer process attribute stores the name of the reviewer, the default value is "ProcessReviewer" or the value assigned in the Human Task configuration. The reviewerType process attribute stores the type of reviewer which can be: user, role or group. You can set these attributes dynamically to modify the effective reviewer.

- 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 How to Configure the Error Assignee and Reviewers.

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 Escalating_ Renewing_ or Ending the Task.

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, 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 below by email when a task assignee requests additional information before they can act upon a task:

```
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 added to the bad address list.

For more information about notifications, see the following:

- Using the Notification Service
- Specifying Participant Notification Preferences
- Part XI, "Using Oracle User Messaging Service"

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 navigate 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.

For more information, see the following:

- Designing Task Forms for Human Tasks .
- Using

Advanced Concepts

This section describes advanced human workflow concepts.

- 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 How to Specify Advanced Task Routing Using Business Rules.

- 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 Assigning Task Participants.

- 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 Assigning Task Participants.

- 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 Introduction to Access Rules.

- 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 Specifying Java Callbacks.

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 Using .

Introduction to the Stages of Human Workflow Design

Human workflow modeling consists of three stages of modeling:

- Stage 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. For more information, see Introduction to Creating a Human Task Definition.
- Stage 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. For more information, see Introduction to Associating the Human Task Definition with a BPEL Process.

- Stage 3: You create a task form. This form displays the task details on which you act at runtime in Oracle BPM Worklist. For more information, see Introduction to Generating the Task Form.

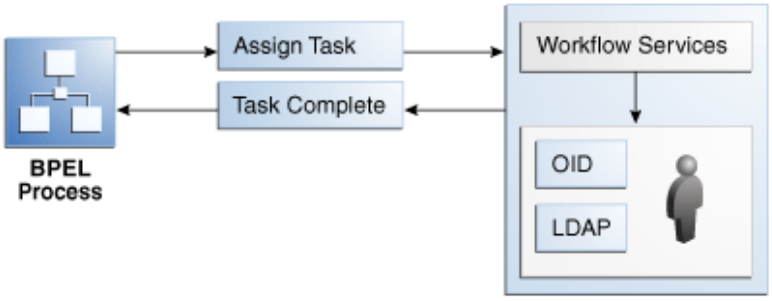
Introduction to Human Workflow Use Cases

This section provides an introduction to use cases for human workflow and services. After that, a tutorial guides you through the design of a human task from start to finish.

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 27-3.

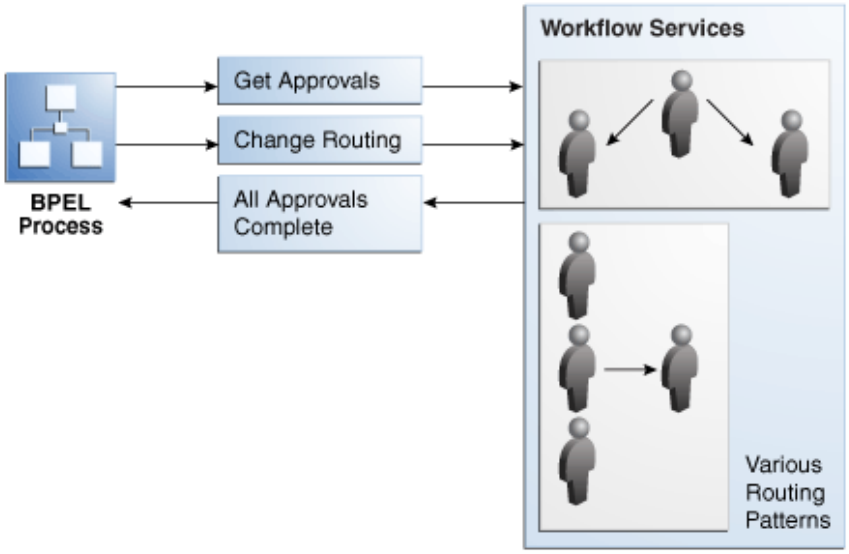
Figure 27-3 Assigning Tasks to a User or Role from a Directory



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 27-4.

Figure 27-4 Flow Patterns and Routing Policies

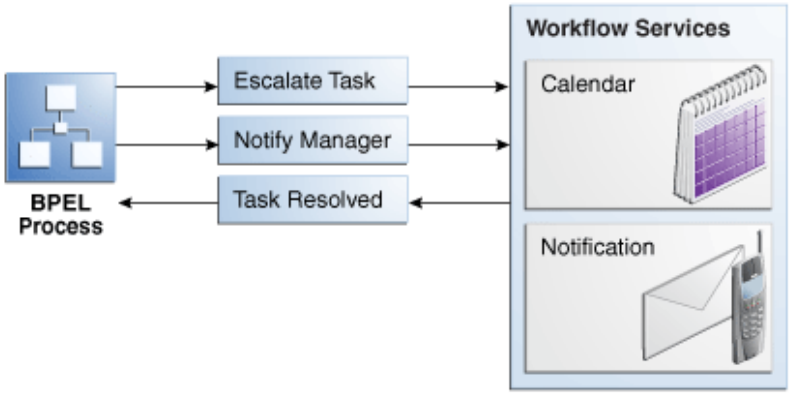


You can use these types as building blocks to create complex workflows.

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 27-5. 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 27-5 Escalation and Notification



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).

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.

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

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 27-6 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 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 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 27-6 Workflow Services Components

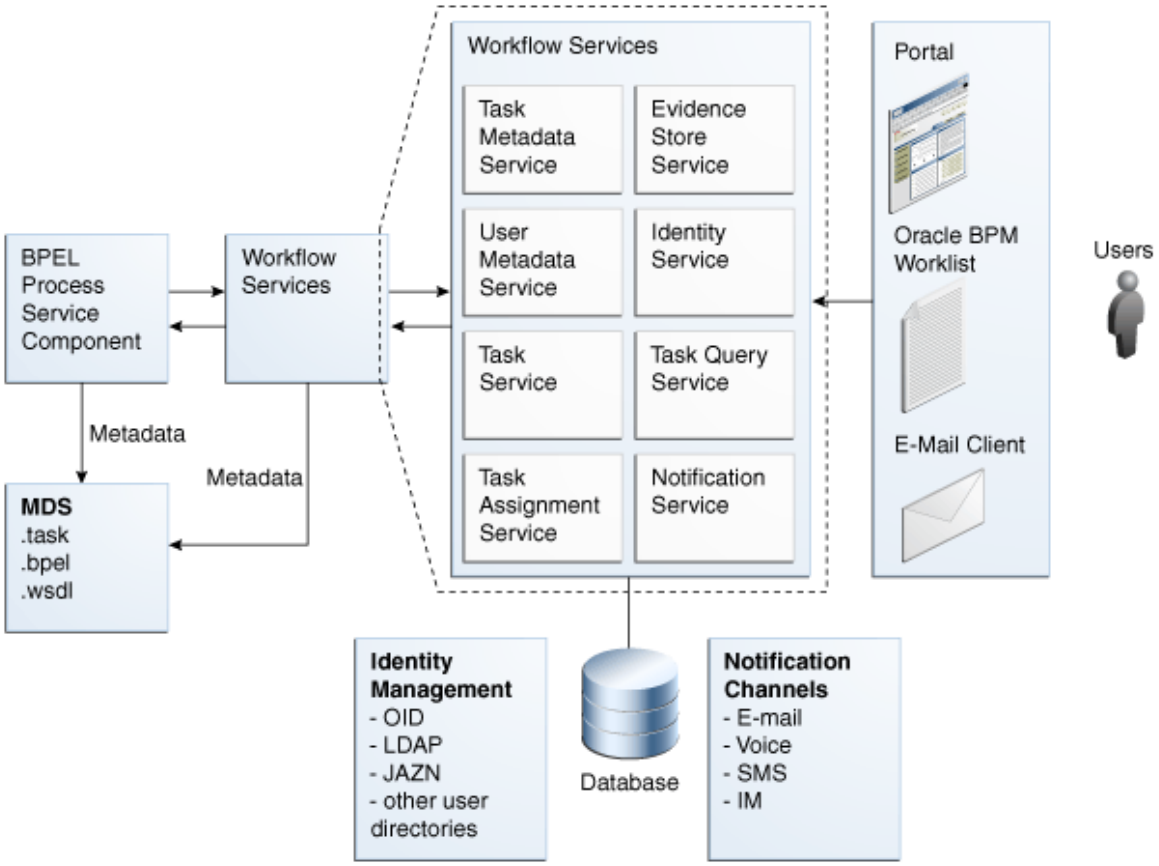
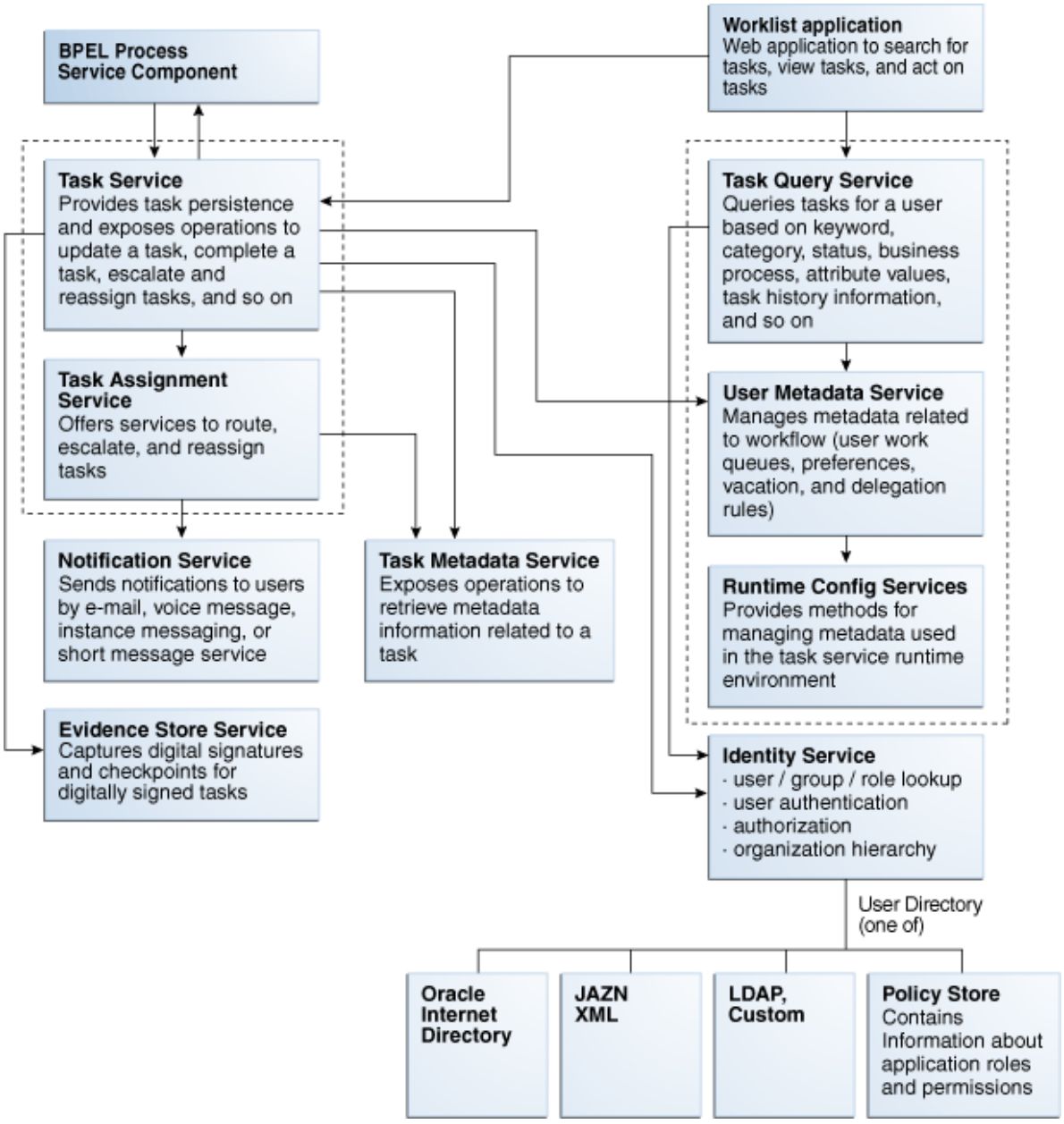


Figure 27-7 shows the interactions between the services and the business process.

Figure 27-7 Workflow Services and Business Process Interactions



Use of Human Task

You can use a human task in the following ways:

- Human task associated with a BPEL process

  You can 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.

- Human task associated with a BPMN process

  You can associate your human task with a BPMN process. The BPMN process may contain other types of BPMN flow objects as part of the flow of the process. The human task is the implementation of a BPMN user task.

- Standalone human task

  You can also create the human task as a standalone component only in the 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.

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 *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

Human Workflow and Business Rule Differences Between Oracle SOA Suite and Oracle BPM Suite

Oracle SOA Suite and Oracle Business Process Management (BPM) Suite both provide support for business rules and human workflow. However, Oracle BPM Suite provides additional business rules and human workflow features that are not available in Oracle SOA Suite. Table 27-1 identifies which business rule and human workflow features are supported in each suite.

Table 27-1 Business Rule and Human Workflow Features in Oracle SOA Suite and Oracle BPM Suite

Feature	Supported in Oracle BPM Suite?	Supported in Oracle SOA Suite?
Workspaces, process tracking, standard dashboards, case management, and applications menu	Yes	No
Approval groups (participant list)	Yes	No
Human workflow and business rules (participant list, routing rules)	Yes	Yes
Verbal rules	Yes	No
Rules business phrases	Yes	No
Oracle BPM Composer - design time rules editing	Yes	No
Process asset catalog (PAM) for source management between Oracle BPM Studio and Oracle BPM Composer	Yes	No
Rules testing in both Oracle JDeveloper and SOA Composer with usability enhancements	Yes	Yes

Table 27-1 (Cont.) Business Rule and Human Workflow Features in Oracle SOA Suite and Oracle BPM Suite

Feature	Supported in Oracle BPM Suite?	Supported in Oracle SOA Suite?
Microsoft Excel import/export for rules decision tables	Yes	Yes

For more information about Oracle BPM Suite, see *Developing Business Processes with Oracle Business Process Management Studio*.

28

Creating Human Tasks

This chapter describes how to create a human task, save it, and associate it with a BPEL process. It also describes how to delete a human task and remove its association with a BPEL process.

This chapter includes the following sections:

For information about human task concepts, see Getting Started with Human Workflow .

For information about troubleshooting human workflow issues, see section "Human Workflow Troubleshooting" of *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

For information about installing and using the organizational hierarchy of users and groups known as the demo user community, see Appendix "Installing the Demo User Community in the Database" of *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

Introduction to Human Tasks

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
- Associating it with a BPEL process
- Generating the task form for displaying the human task during runtime in Oracle BPM Worklist.

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

This section provides a brief overview of these modeling tasks and provides references to specific modeling instructions.

For more information about using the , see Getting Started with Developing SOA Composite Applications.

For information about available samples, see Human Workflow Tutorial.

Introduction to Creating a Human Task Definition

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. 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.

After you create a Human Task you can configure its metadata using the Human Task Editor. For a detailed description of the metadata and configuration procedures, see Configuring Human Tasks .

You define the metadata for the human task in either of two ways:

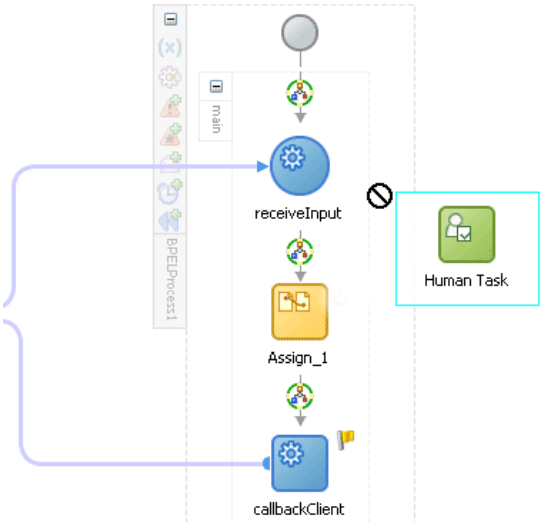
- By dragging a human task from the Components window into a BPEL process in Oracle BPEL Designer and clicking the **Add** icon in the Create Human Task dialog that automatically is displayed. This displays a dialog for creating the human task service component. When creation is complete, the Human Task Editor is displayed.
- By dragging a human task service component from the Components window 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.

For more information, see Creating Human Tasks.

Introduction to Associating 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 28-1.

Figure 28-1 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 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 Associating Human Tasks with BPEL Processes.

Introduction to Generating 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 Designing Task Forms for Human Tasks .

Creating Human Tasks

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.

You create a human task service component in the 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 .

How to Create a Human Task Using the SOA Composite Editor

You can create a human task using the SOA Composite Editor. You can use this method to create a human task to later associate with a BPEL process or use as a standalone component.

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 .
2. From the Components window list, select **SOA**.

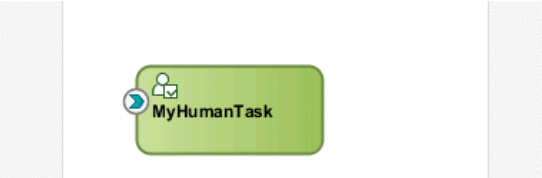
   The list refreshes to display service components and service adapters.
3. From the list, drag a **Human Task** into the designer.

   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** check box. The selection of this check box determines how the human task service component is created.

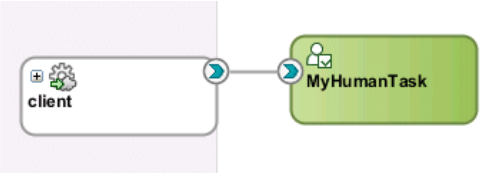
   a. To create a human task service component that you later associate with a BPEL process service component, do *not* select the **Create Composite Service with SOAP Bindings** check box. The human task service component is created as a component that you explicitly associate with a BPEL process service component. Figure 28-2 provides details.

   Figure 28-2 Human Task Component

   b. To create the human task service component as a standalone component in the , select the **Create Composite Service with SOAP Bindings** check box. This creates a human task service component that is automatically wired to a Simple Object Access Protocol (SOAP) web service. Figure 28-3 provides details.

   Figure 28-3 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**.

For more information about creating a human task service component in the , see Getting Started with Developing SOA Composite Applications.

How to Create a Human Task Using Oracle BPEL Designer

You can create a human task using Oracle BPEL Designer. Generally you use this method when you want to create a human task to use with a BPEL process.

To create a human task in Oracle BPEL Designer:

1. In the Components window, 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**.

What Happens When You Create a Human Task

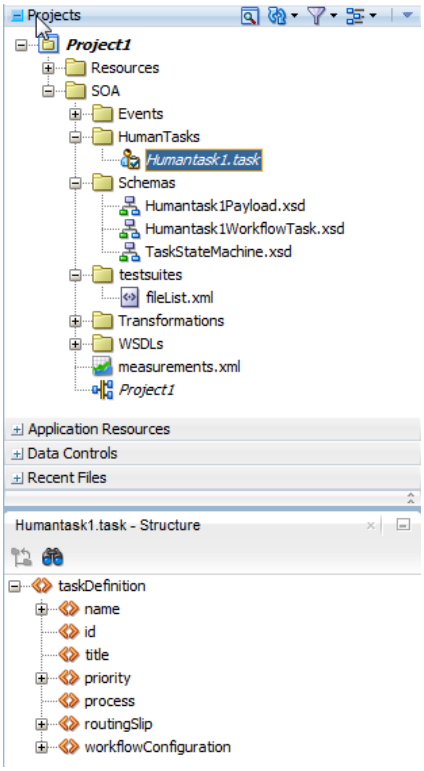
When a human task is created, the following folders and files appear:

- 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 Applications window under ***SOA_Project_Name*** **> SOA**. You can re-edit the settings in this file by double-clicking the following:
  - The `.task` file in the Applications window in either the or Oracle BPEL Designer
  - The human task icon in the 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 .

Figure 28-4 shows these folders and files.

Figure 28-4 Human Task Folders and Files



For information about available samples, see Human Workflow Tutorial.

Configuring Human Tasks

After you create a human task, you modify its settings using the Human Task Editor. For more information on how to configure a human task, see Configuring Human Tasks .

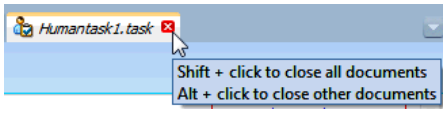
Exiting the Human Task Editor and Saving Your Changes

You can save your human task changes at any time. The task can be re-edited at a later time by double-clicking the metadata task configuration `.task` file in the Applications window.

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 28-5 to close the `.task` metadata task configuration file.

Figure 28-5 File Closure



2. If you click the **X** sign, select **Yes** when prompted to save your changes.

Associating Human Tasks with BPEL Processes

To associate the human task service component created in the 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 Creating Human Tasks.

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:

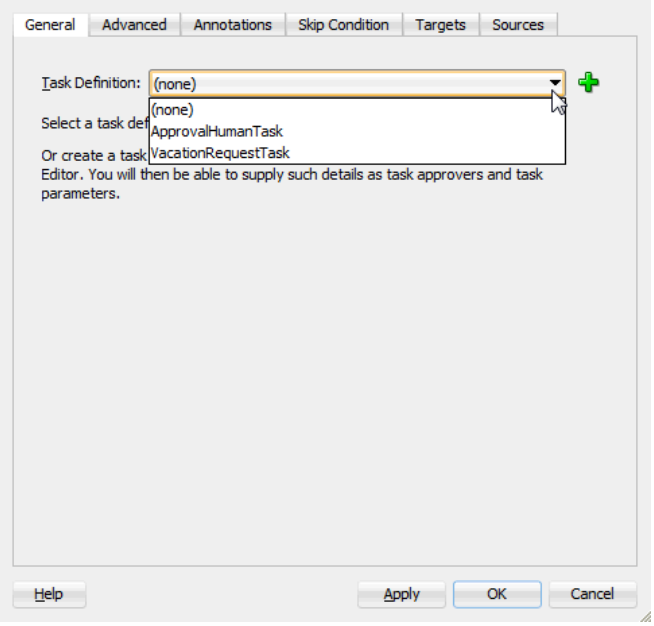
- If you have created a human task service component in the SOA composite application, drag a human task activity into the BPEL process in Oracle BPEL Designer. Then, select the existing human task service component from the **Task Definition** list of the Create Human Task dialog. You can then specify 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 .
2. Double-click the BPEL process service component with which to associate the `.task` file of the human task service component.
3. In the Components window, 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 28-6.

Figure 28-6 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.

What You May Need to Know About Deleting a Wire Between a Human Task 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:

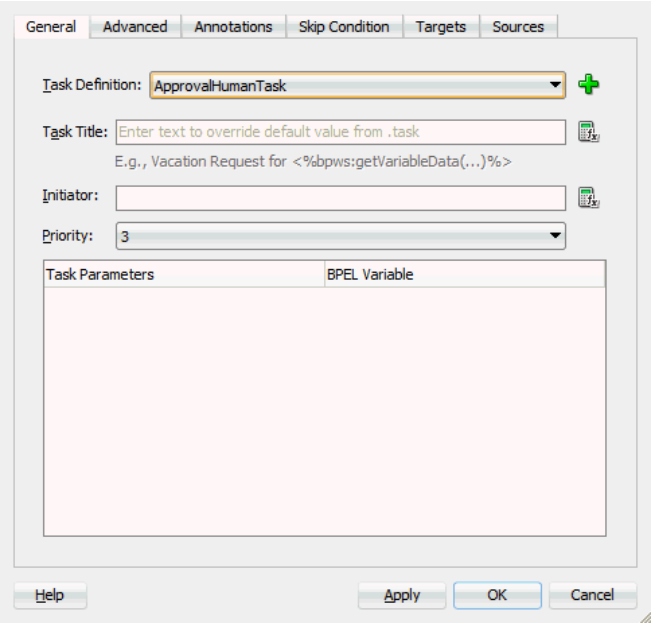
- The switch activity contains user-entered BPEL code.
- The switch can be reused if the intention for deleting the wire is only to point to another human task.
- Deleting the switch is a single-step action.

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.

How to Define the Human Task Activity Title, Initiator, Priority, and Parameter Variables

Figure 28-7 shows the **General** tab that displays after you select the human task.

Figure 28-7 Human Task — General Tab (After Selection)



The **General** tab of the Human Task activity enables you to perform the tasks shown in Table 28-1:

Table 28-1 Human Task - General Tab

For this Field...	See...
Task Title	Specifying the Task Title
Initiator	Specifying the Task Initiator and Task Priority
Priority	
Task Parameters	Specifying Task Parameters

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 How to Specify a Task Title.

In the **Task Title** field of the **General** tab, enter the task title by entering the title manually. Alternatively, 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.

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.

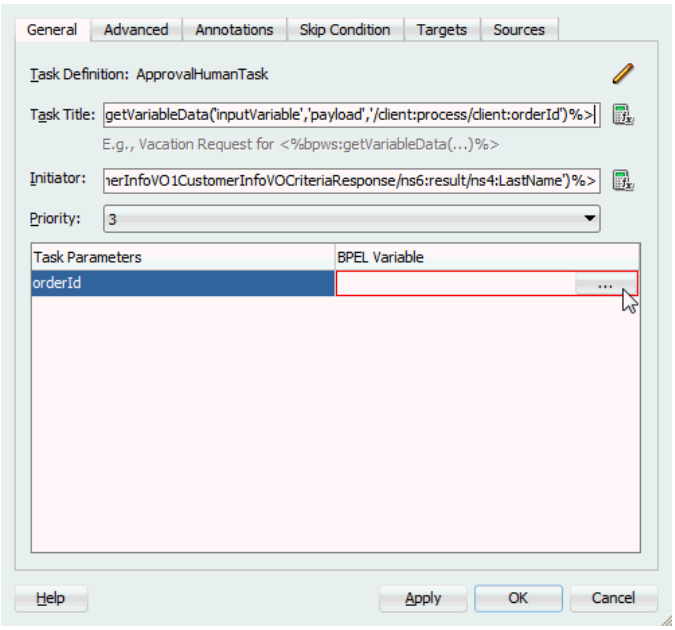
2. From the **Priority** list, select a priority value between **1** (the highest) and **5**. This field is provided for user reference and does not make this task a higher priority during runtime. Use the priority to sort tasks in Oracle BPM Worklist. This priority value overrides the priority value 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 How to Specify a Task Title.

Specifying Task Parameters

The task parameter table shown in Figure 28-8 displays a list of task parameters after you complete the **Task Title** and **Initiator** fields.

Figure 28-8 Task Parameter Table



To specify task parameters:

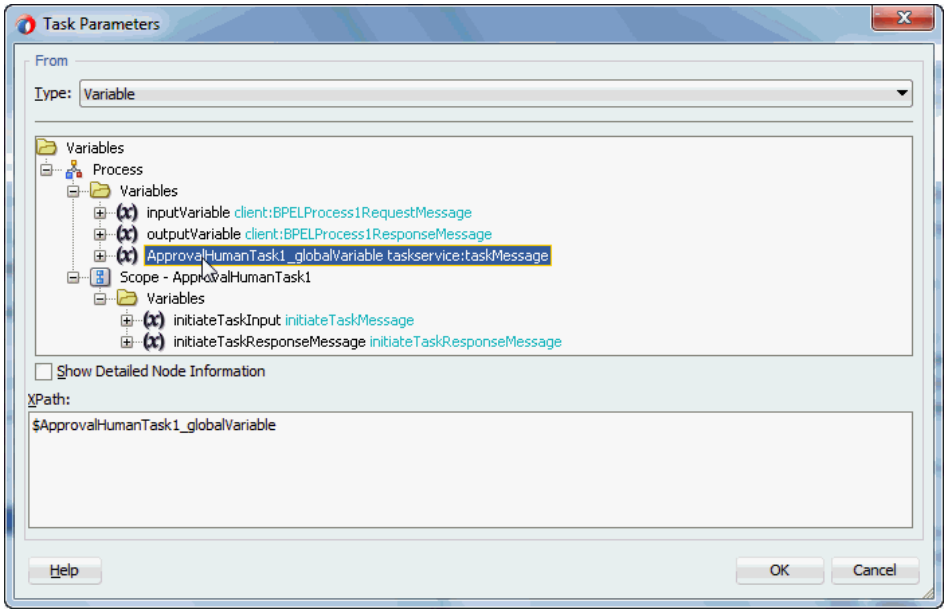
1. In the **BPEL Variable** column, double-click the **dots** to map the task parameter to the BPEL variable. To display these dots for selection, you must have already specify your data parameters. For more information on how to specify the data

parameters, see How to Specify the Task Payload Data Structure. 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 28-9 and select the appropriate task variable.

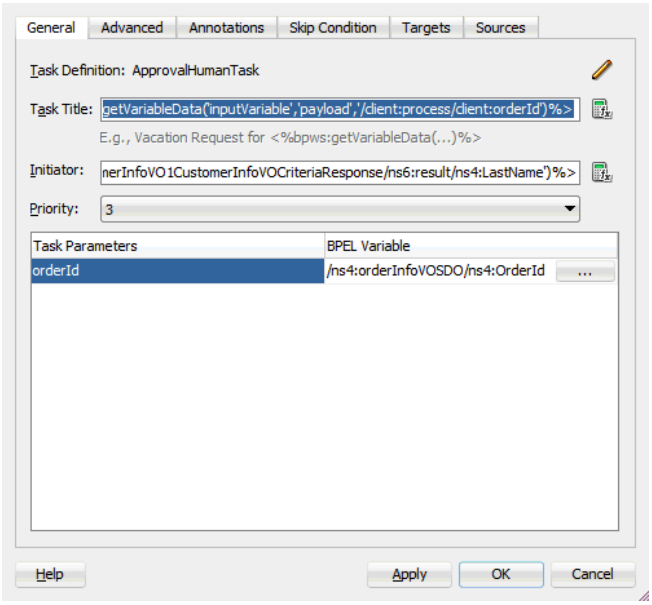
Figure 28-9 Variables Tree



3. Click **OK**.

The Human Task dialog shown in Figure 28-10 appears as follows.

Figure 28-10 Human Task Dialog

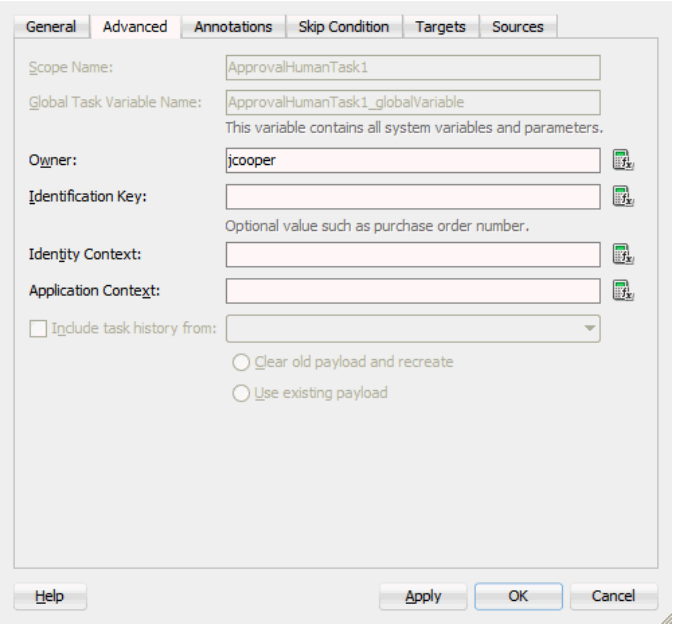


4. To define advanced features for the human task activity, click the **Advanced** tab and go to How to Define the Human Task Activity Advanced Features . Otherwise, click **OK** to close the Human Task dialog.

How to Define the Human Task Activity Advanced Features

Figure 28-11 shows the **Advanced** tab.

Figure 28-11 Create Human Task — Advanced Tab



The **Advanced** tab of the Human Task activity enables you to perform the tasks shown in Table 28-2:

Table 28-2 Human Task - Advanced Tab

For this Field...	See...
Scope Name **Global Task Variable Name**	Specifying a Scope Name and a Global Task Variable Name
Owner	Specifying a Task Owner
Identification Key	Specifying an Identification Key
Identity Context	Specifying an Identity Context
Application Context	Specifying an Application Context
Include task history from	Including the Task History of Other Human Tasks

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.

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. 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.

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.

In the **Identification Key** field of the **Advanced** tab, enter an optional identification key value to specify a key.

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, in the **Identity Context** field of the **Advanced** tab, enter a value

Specifying an Application Context

The stripe name of the application contains the application roles used in the task. To specify an application context, in the **Application Context** field of the **Advanced** tab, enter a value.

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** check box of the **Advanced** tab to extend a previous workflow task in the BPEL process. Selecting this check box includes the task history, comments, and attachments from the previous task. This provides you with a complete end-to-end audit trail.

   When a human task is continued with another human task, the following information is carried over to the new workflow:

   - Task payload and the changes made to the payload in the previous workflow
   - Task history
   - Comments added to the task in the previous workflow
   - Attachments added to the task in the previous workflow
   - 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**

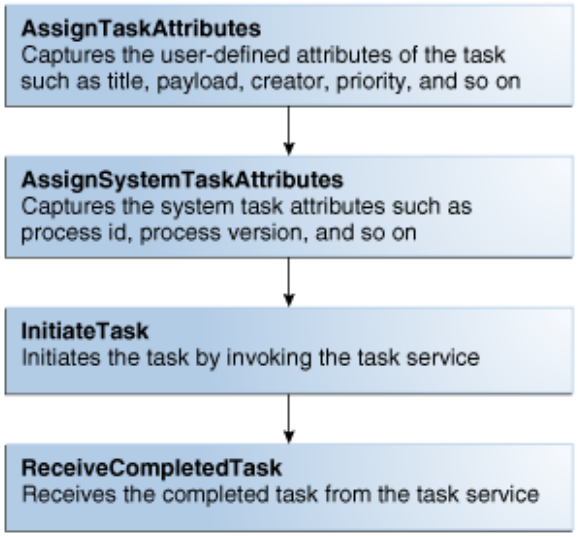
     This option is applicable when the payload attributes in the XML files of the human tasks involved in this extended workflow are the same.

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 28-12 shows how a workflow interaction is modeled. Figure 28-12 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 28-12 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 28-13.

Figure 28-13 Expanding the Human Task Activity

Invoking BPEL Callbacks

If intermediary events must be propagated to the BPEL process instance, select the **Allow task and routing customization in BPEL callbacks** check box 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`

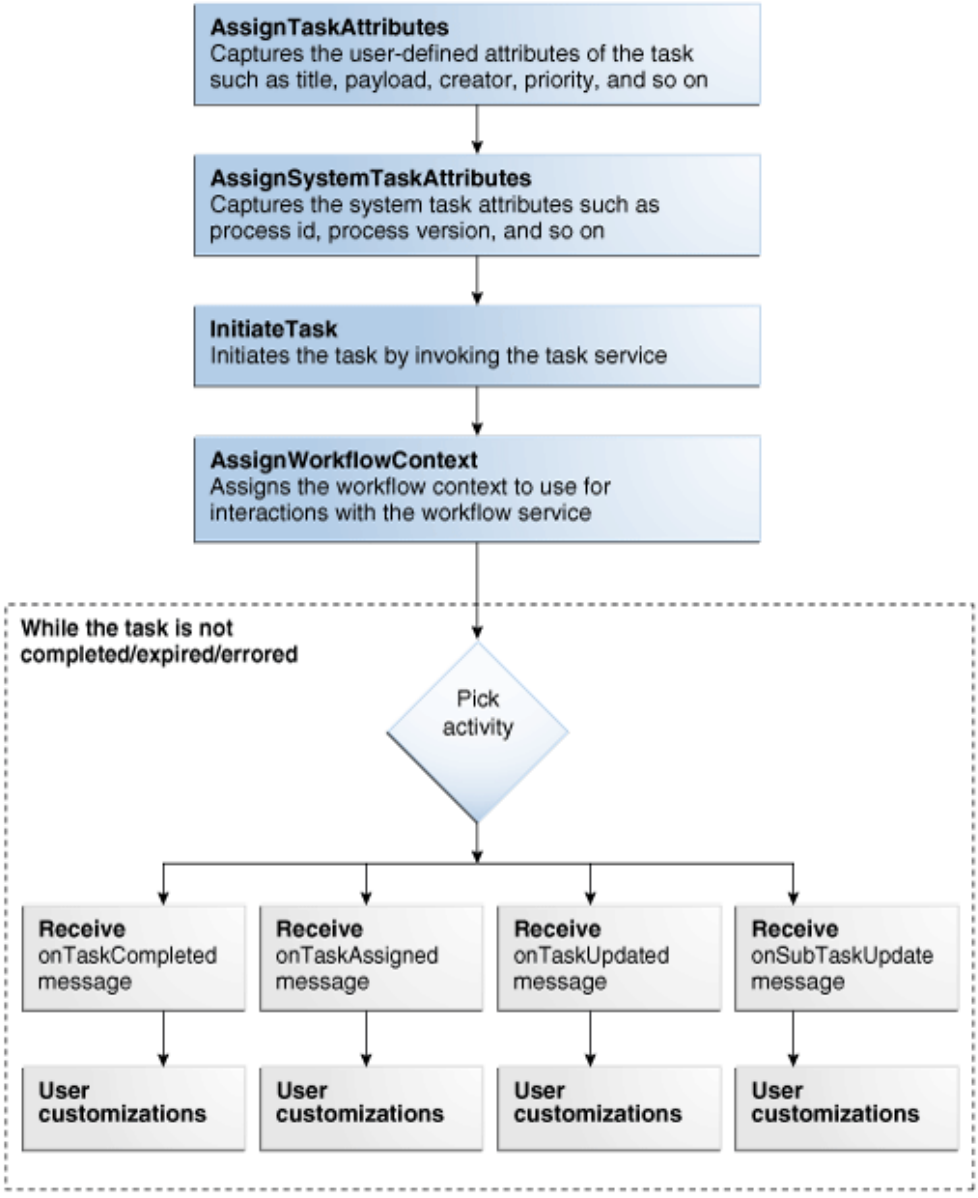
  This callback is invoked for any update to a subtask.

Figure 28-14 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 28-14 Workflow Interaction Modeling (with Callbacks)



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.
- When you copy comments to a human task, make sure that those comments do not contain the task ID. The taskId element must be empty.

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.

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.

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 you can remove the assign generated in the switch-case after the task scope.

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 the following example:

```
<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>
```

What You May Need to Know About Encoding an Attachment

To enable text files to be attached to a human task, you must set a flag that describes whether the content of text attachments is encoded. This flag is named `isContentEncoded`.You can set this flag by customizing the BPEL code in any Human Workflow sample that includes a human task. To do this customization, in the `.bpel` file in the sample, enter the following copy rule in the BPEL assign activity code:

```
<copy>
<from>true()</from>
<to>$initiateTaskInput.payload/task:task/task:attachment/task:isContentEncoded
</to>
</copy>
```

Once you have entered this copy rule, you can either save the file and continue designing the BPEL process or, if you have finished designing, you can deploy the process.

29

Configuring Human Tasks

This chapter describes how to configure the different properties of a human task. It covers basic properties, task payload data structure, participant assignment, routing policies, localization, escalation, notification preferences, access policies and task actions, restrictions and Java and business event callbacks.

This chapter includes the following sections:

For information about troubleshooting human workflow issues, see section “Human Workflow Troubleshooting" of *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

Accessing the Sections of the Human Task Editor

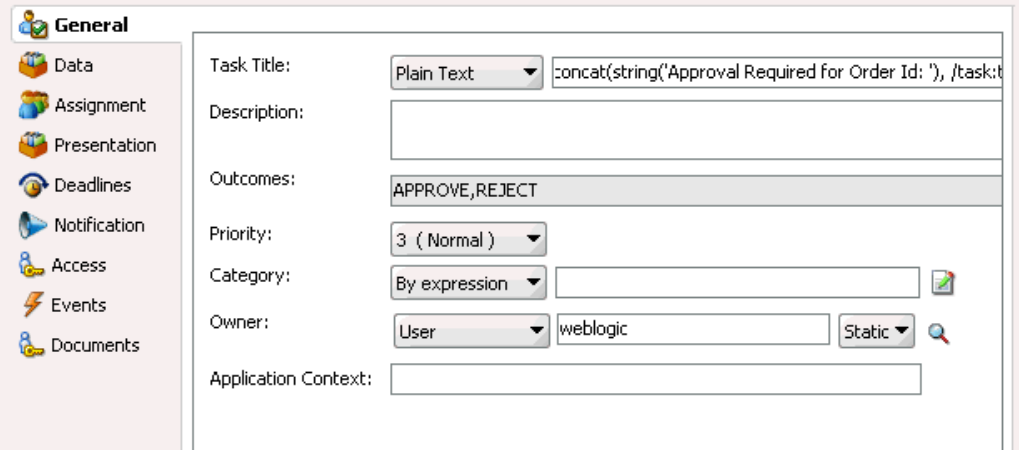
This section describes how to access the sections of the Human Task Editor. Brief descriptions are provided of each section and references are provided to more specific information.

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.

   The Human Task Editor consists of the main sections shown on the left side in Figure 29-1. These sections enable you to design the metadata of a human task.

Figure 29-1 Human Task Editor



Instructions for using these main sections of the Human Task Editor to create a workflow task are listed in Table 29-1.

Table 29-1 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.	Specifying 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).	Specifying 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.	Assigning Task Participants Selecting a Routing Policy
Presentation	Enables you to specify the following settings: • Multilingual settings • WordML and custom style sheets for attachments	Specifying Multilingual Settings and Style Sheets
Deadlines	Enables you to specify the expiration duration of a task, custom escalation Java classes, and due dates.	Escalating_ Renewing_ or Ending 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.	Specifying Participant Notification Preferences

Table 29-1 (Cont.) Human Task Editor

Section	Description	See...
Access	Enables you to specify access rules for task content and task actions, workflow signature policies, and assignment restrictions.	Specifying Access Policies and Task Actions on Task Content How to Specify a Workflow Digital Signature Policy Specifying Restrictions on Task Assignments
Events	Enables you to specify callback classes and task and routing assignments in BPEL callbacks.	Specifying Java or Business Event Callbacks

Specifying the Title, Description, Outcome, Priority, Category, Owner, and Application Context

This section explains how to specify the task details such as the title, description, outcome, priority, category Owner and the Application context.

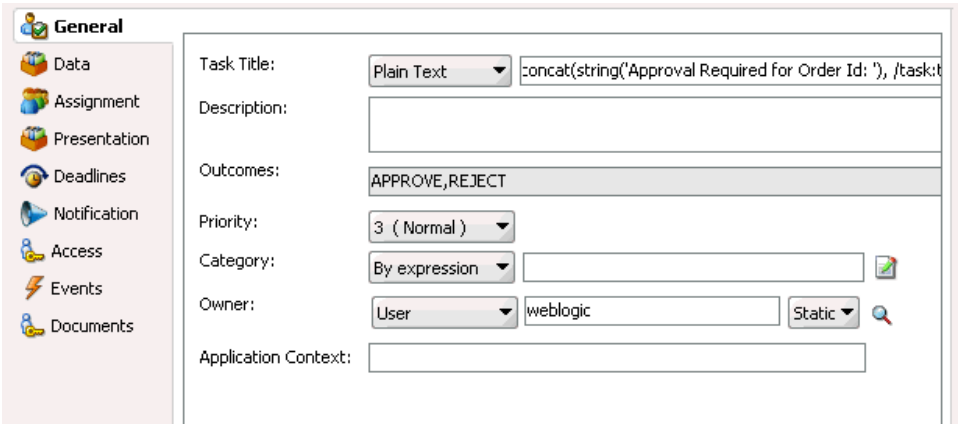
To specify the details of a task:

1. Access the Human Task Editor.
2. Click the **General** tab.

   Figure 29-2 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 29-2 Human Task Editor — General Section



Instructions for configuring the following subsections of the **General** section are listed in Table 29-2:

Table 29-2 Human Task Editor — General Section

For This Subsection...	See...
Title	How to Specify a Task Title

Table 29-2 (Cont.) Human Task Editor — General Section

For This Subsection...	See...
Description	How to Specify a Task Description
Outcomes	How to Specify a Task Outcome
Priority	How to Specify a Task Priority
Category	How to Specify a Task Category
Owner	How to Specify a Task Owner
Application Context	How To Specify an Application Context

How 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.

To specify a task title:

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.
   - **Translation**: Click the Lookup button and locate a translation bundle to use to specify the title.
   - **Resource Xpath**: Click the Lookup button and locate a resource bundle to use to specify the title.

2. If you enter a title in the **Task Title** field of the **General** tab of the Create Human Task dialog box described in Specifying the Task Title, the title you enter here is overridden.

How to Specify 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.

To add a task description:

1. Select the drop-down menu and choose either **Plain Text** or **Translation**.
2. Provide the description:

   Plain text:

   a. Type a description into the dialog box.

   b. Click **Ok**.

   Translation:

   a. Click the **Lookup** button.

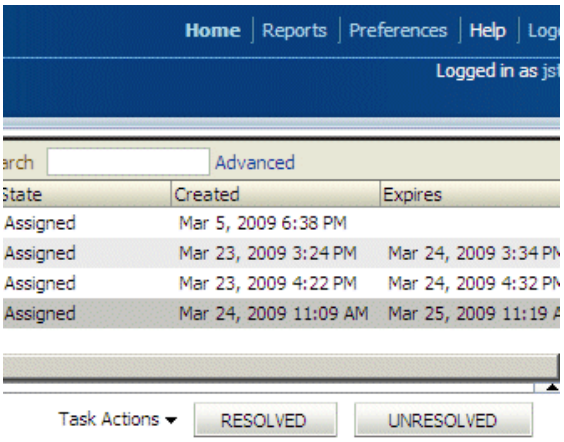
   b. Locate a resource bundle and provide a description.

   c. Click **Ok**.

How to Specify 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 29-3 provides details.

Figure 29-3 Outcomes in Oracle BPM Worklist



You can specify the following types of task outcomes:

- Select a seeded outcome
- Enter a custom outcome

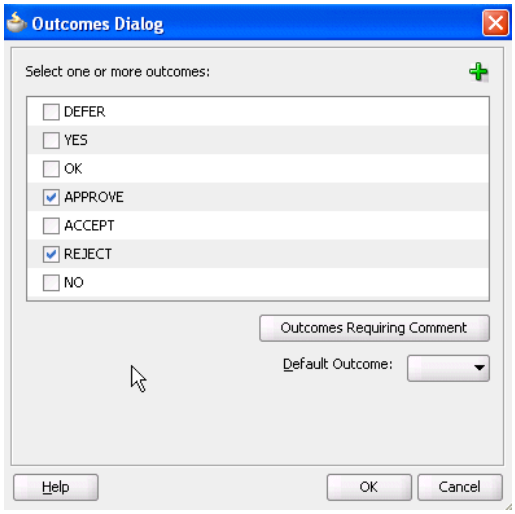
The task outcomes can also have runtime display values that are different from the actual outcome value specified here. This permits outcomes to be displayed in a different language in Oracle BPM Worklist. For more information about internationalization, see How to Specify Multilingual Settings.

To specify a task outcome:

1. To the right of the **Outcomes** field in the **General** section, click the **Search** icon.

   The Outcomes dialog box shown in Figure 29-4 displays the possible outcomes for tasks. **APPROVE** and **REJECT** are selected by default.

Figure 29-4 Outcomes Dialog



2. Enter the information shown in Table 29-3.

Table 29-3 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 box for adding custom outcomes. In the **Name** field of the dialog box, 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 must specify an outcome with spaces, use a resource bundle. For more information, see 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.

Table 29-3 (Cont.) Outcomes Dialog

Field	Description
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 Specifying the Voting Outcome.

How to Specify 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 box. You can filter tasks based on priority and create views on priorities in Oracle BPM Worklist.

From the **Priority** list in the **General** section, select a priority for the task to specify a priority.

For more information about specifying a priority value in the Create Human Task dialog box, see Specifying the Task Initiator and Task Priority.

How to Specify 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.
   - **Translation**: If the composite contains a resource bundle file, then use the Lookup button to locate the resource bundle file and to specify a category.

How to Specify 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 box described in 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 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 Task Assignment and Routing.

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.

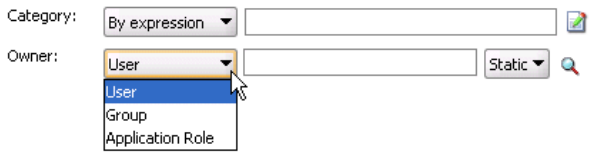
To specify a task owner statically through the user directory or a list of application roles:

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 29-5 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 *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

   Figure 29-5 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**.

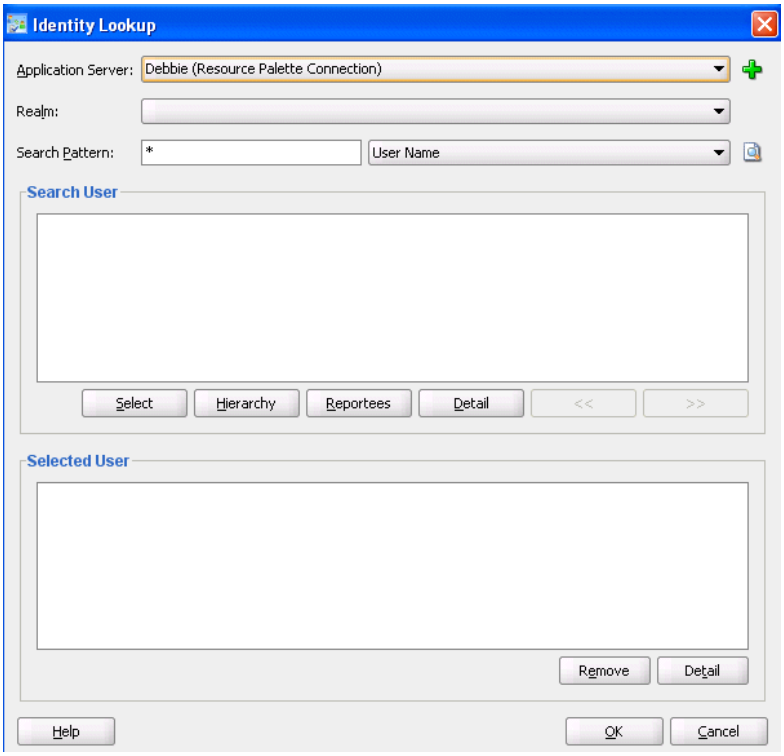
3. See the step in Table 29-4 based on the type of owner you selected.

Table 29-4 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 box shown in Figure 29-6 appears.

Figure 29-6 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 box. Instead, create an application server connection to a managed Oracle WebLogic Server.

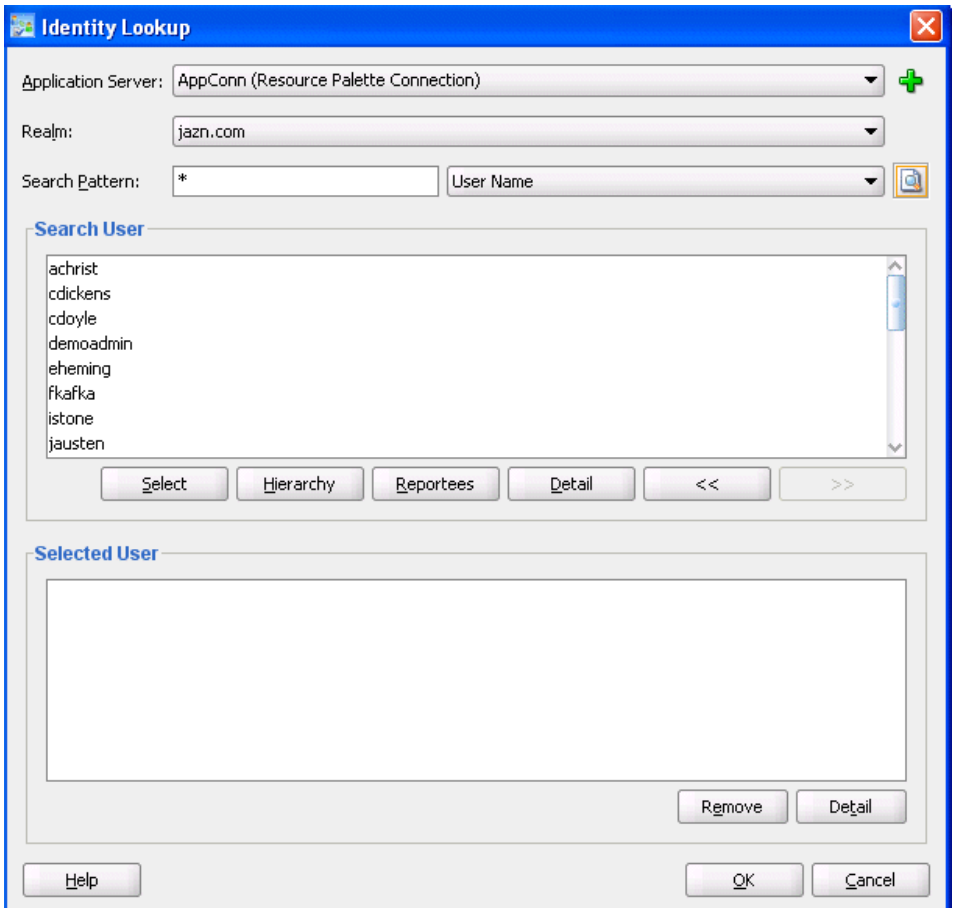
- You must select an application server connection configured with the complete domain name (for example, `myhost.us.example.com`). If you select a connection configured only with the hostname (for example, `myhost`), the **Realm** list may not display the available realms. If the existing connection 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 box 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).

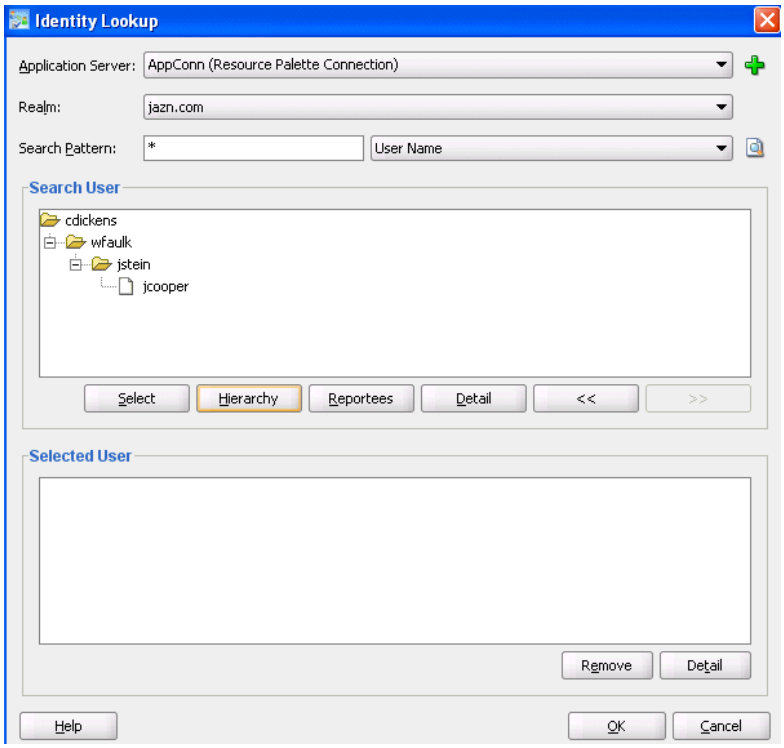
b. Search for the owner by entering a search string such as `jcooper`, `j*`, `*`, and so on. Clicking the **Lookup** icon to the right of the **User Name** field fetches all the users that match the search criteria. Figure 29-7 provides details. One or more users or groups can be highlighted and selected by clicking **Select**.

Figure 29-7 Identity Lookup with Realm Selected



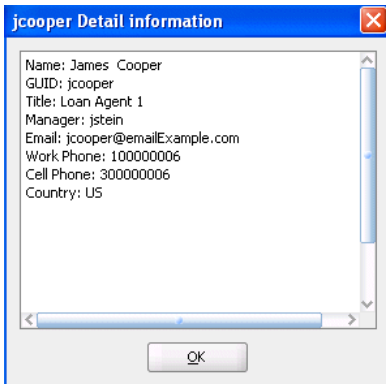
c. View the hierarchy of a user by highlighting the user and clicking **Hierarchy**. Similarly, clicking **Reportees** displays the reportees of a selected user or group. Figure 29-8 provides details.

Figure 29-8 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 29-9 provides details.

Figure 29-9 User or Group Details



e. Click **OK** to return to the Identity Lookup dialog box.

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.

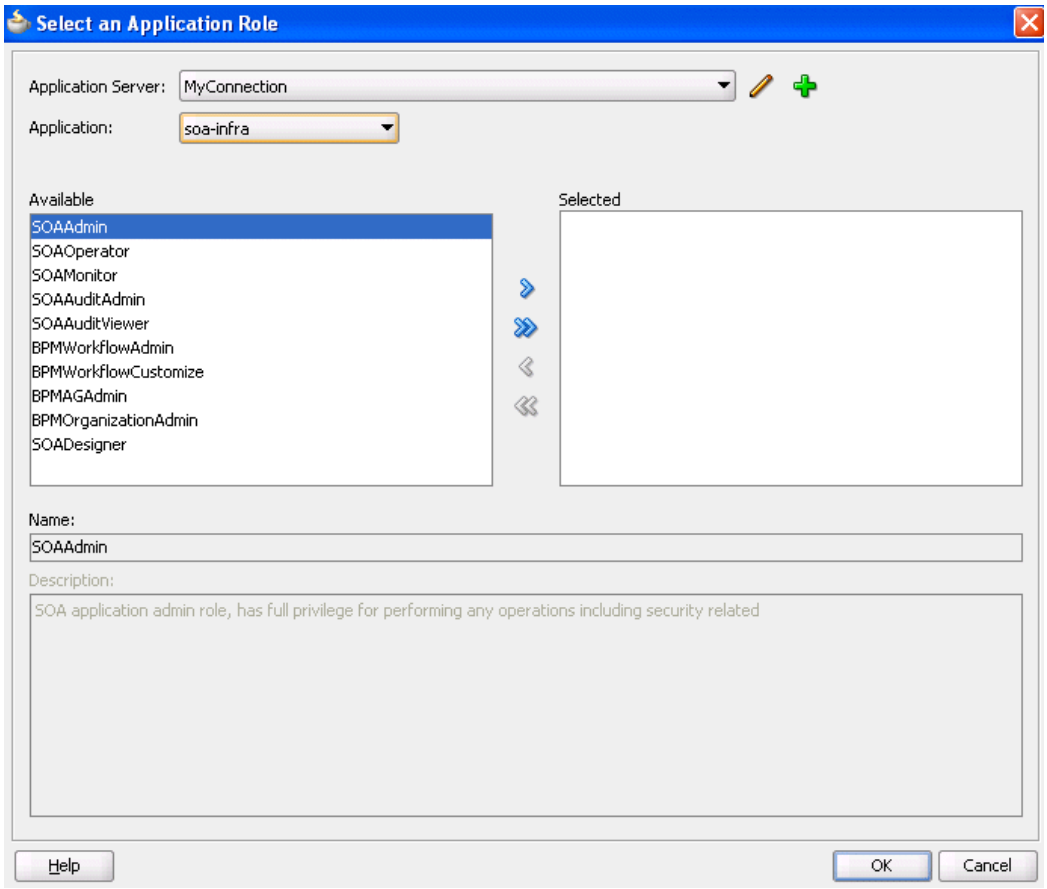
5. If you selected **Application Role**, the Select an Application Role dialog box appears.

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 29-10 provides details.

Figure 29-10 Application Role



d. Click **OK**.

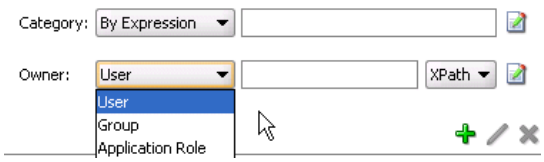
Specifying a Task Owner Dynamically Through an XPath Expression

Task owners can be selected dynamically in the Expression Builder dialog box.

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 29-11 provides details.

Figure 29-11 Specify a Task Owner Dynamically

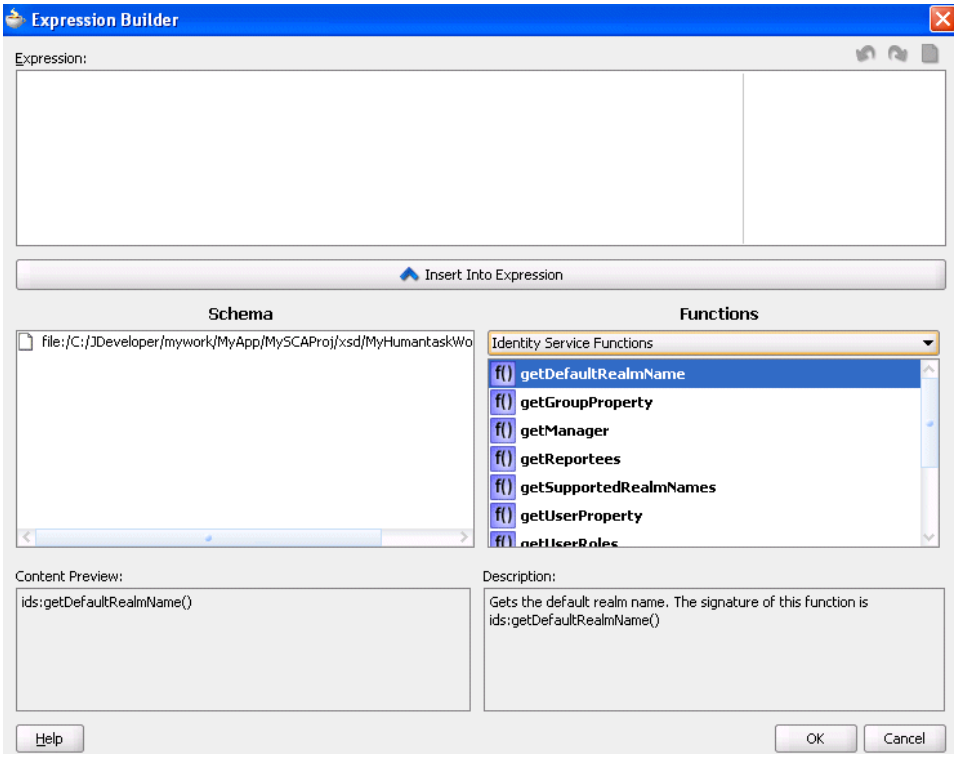


2. In the second list to the right of the **Owner** field in the **General** section, select **XPath**.

3. Click the icon to launch the Expression Builder.

   This displays the Expression Builder dialog box shown in Figure 29-12:

Figure 29-12 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 box and XPath Building Assistant
   - XPath Extension Functions for information about workflow service dynamic assignment functions, identity service functions, and instructions on using the XPath Building Assistant

How To Specify 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.

Note:

An application context is required to be set in the task definition in order to be able to reassign the task to an application role in the Oracle Process Workspace and Oracle BPM Worklist applications.

In the **Application Context** field of the **General** section, enter the name to specify an application context.

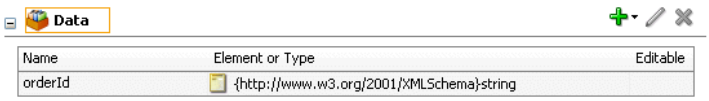
Specifying the Task Payload Data Structure

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.

Figure 29-13 shows the **Data** section of the Human Task Editor.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 29-13 Human Task Editor — Parameters Section



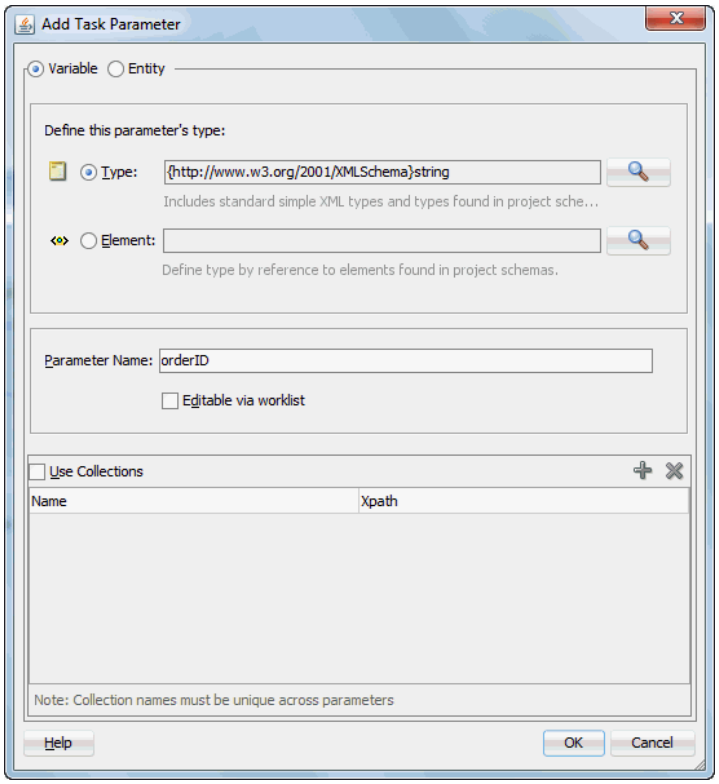
How to Specify the Task Payload Data Structure

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 box is displayed, as shown in Figure 29-14.

Figure 29-14 Add Task Parameter Dialog



3. Enter the details described in Table 29-5:

Table 29-5 Add Task Parameter Dialog Fields and Values

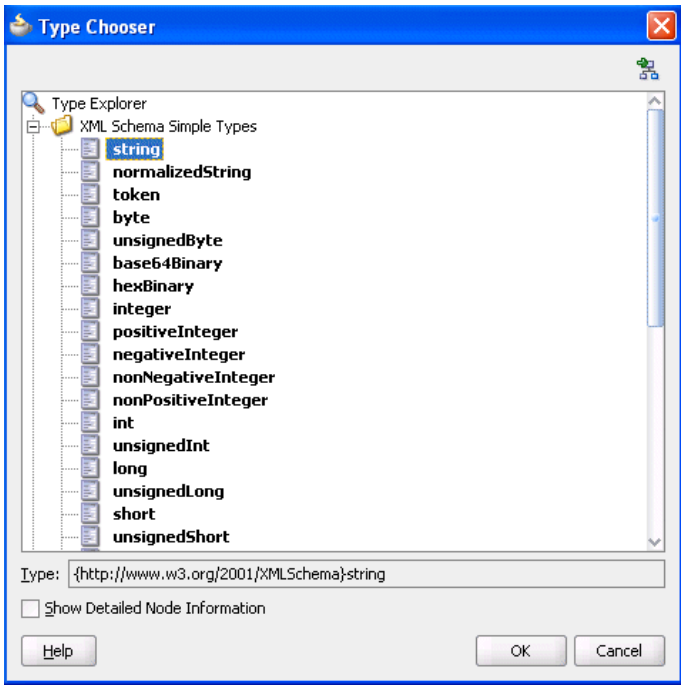
Field	Description
Parameter Type	Select **Type** or **Element** and click the **Search** icon to display the Type Chooser dialog box for selecting the task parameter.
Parameter Name	Accept the default name or enter a custom name. This field only displays if **Type** is the selected parameter type.
Editable via worklist	Select this check box 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 Specifying Access Policies and Task Actions on Task Content.
Use Collections	If a task uses collections, then define this parameter to use collections. Click the **Add** button to provide the collection name and the Xpath expression for the collection type. Use Expression Builder to look up the collection type from the schema.

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 29-15.

Figure 29-15 Parameter Type



5. Click **OK** to return to the Human Task Editor.

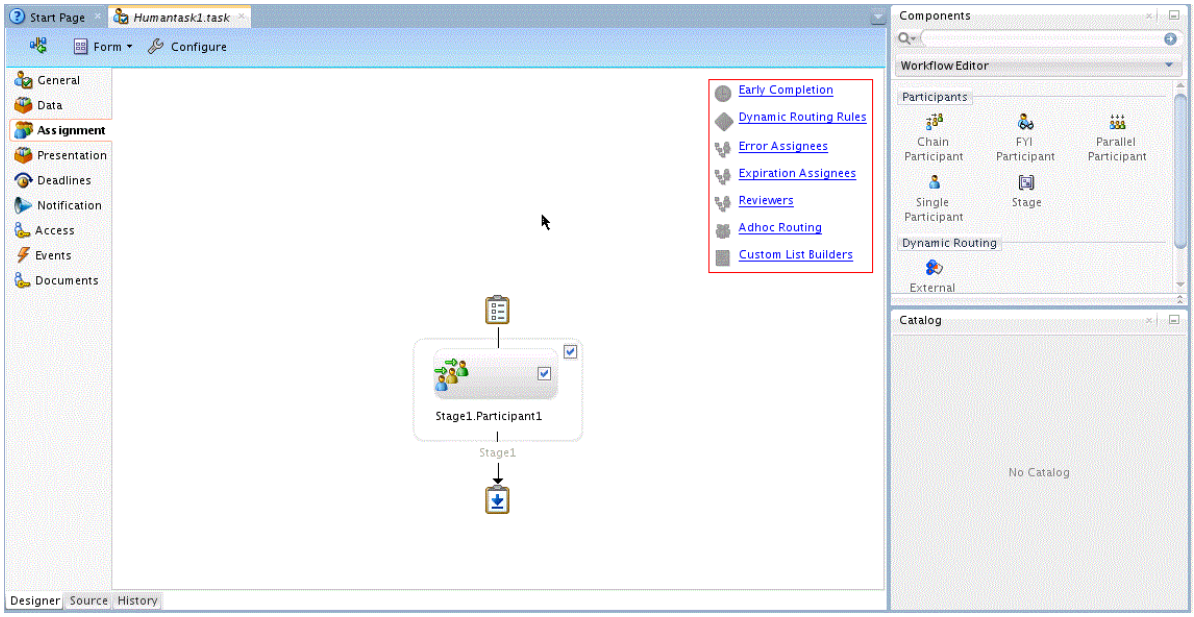
   Your selection displays in the **Data** section.

6. To edit your selection, select it and click the **Edit** icon in the upper right part of the **Data** section.

Assigning Task Participants

Figure 29-16 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 29-16 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 29-16). 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. Creating additional stages is an advanced requirement that may not be necessary for your environment.

For more information about participant types, see Task Assignment and Routing.

How to Specify a Stage Name and Add Parallel and Sequential Blocks

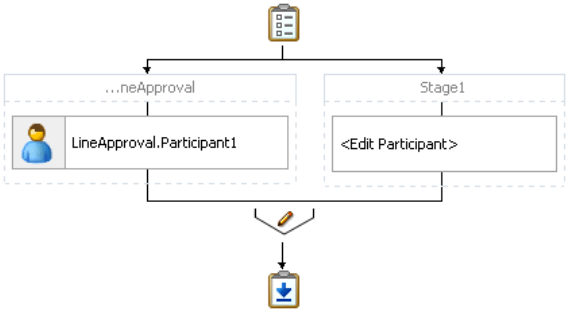
The stage is named **Stage1** by default, however you can change the name.

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

1. Double-click the name. The **Edit** dialog box displays.
2. In the **Edit** dialog box, enter the following details and click **OK**.
   - **Stage**: The name of the stage.
   - **Non Repeating**: Do not stage in parallel for each item in the collection.
   - **Repeat Stage in parallel for each item in a collection**: Choose one collection from the drop-down list to specify which collection type to use for the repeated stages.
3. Drag and drop the type of participant from the **Participant** palette on the right onto the stage.
4. Drag **Stage** from the **Participant Palette** on the right and drop it on the green dot of the existing stage.

   When you bring the new stage closer to the current stage, four green dots display around the current stage. Choose the green dot that is to right to the current stage. A second stage is added in parallel to the first stage, as shown in Figure 29-17.

   Figure 29-17 Parallel Stage

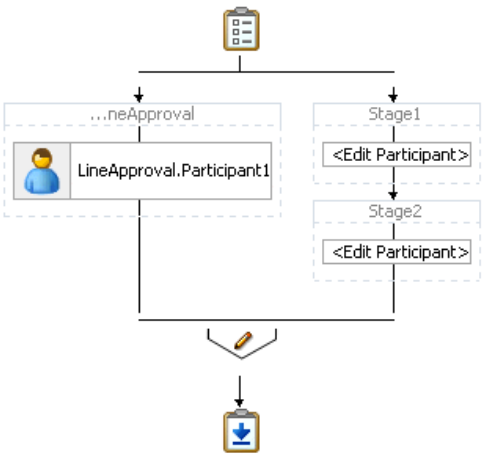


5. Drag **Stage** from the **Participant Palette** on the right and drop it on the green dot of the existing stage

   When you bring the new stage below the current stage, four green dots display around the current stage. Choose the green dot that is below the current stage.

   A sequential stage is added below the selected block.

Figure 29-18 Sequential Stage



You create participant types within these blocks.

How to Assign Task Participants

To assign task participants:

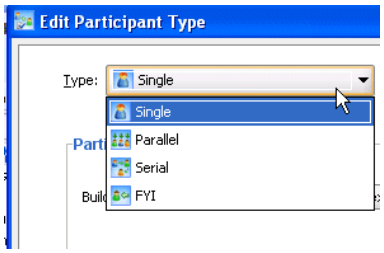
1. In the **Assignment** section, perform one of the following tasks:

   - Drag and drop Participants from the Components window onto Stage. The first time you create a task participant, the box is labeled **<Edit Participant>**.
   - Double-click the participant box.

   The Edit Participant Type dialog box appears. This dialog box enables you to select a specific participant type.

2. From the **Type** list, select a participant type shown in Figure 29-19.

Figure 29-19 Type List



3. See the section shown in Table 29-6 based on your selection.

Table 29-6 Participant Types

Participant Type	For a Description of this Participant Type, See...	For Instructions on Configuring this Participant Type, See...
• Single • Parallel • Serial • FYI	Task Assignment and Routing	How to Configure the Single Participant Type How to Configure the Parallel Participant Type How to Configure the Serial Participant Type How to Configure the FYI Participant Type

How to Configure the Single Participant Type

Figure 29-20 shows the Edit Participant Type dialog box for the single participant type. Figure 29-21 shows the expanded **Advanced** section.

Figure 29-20 Edit Participant Type — Single Type

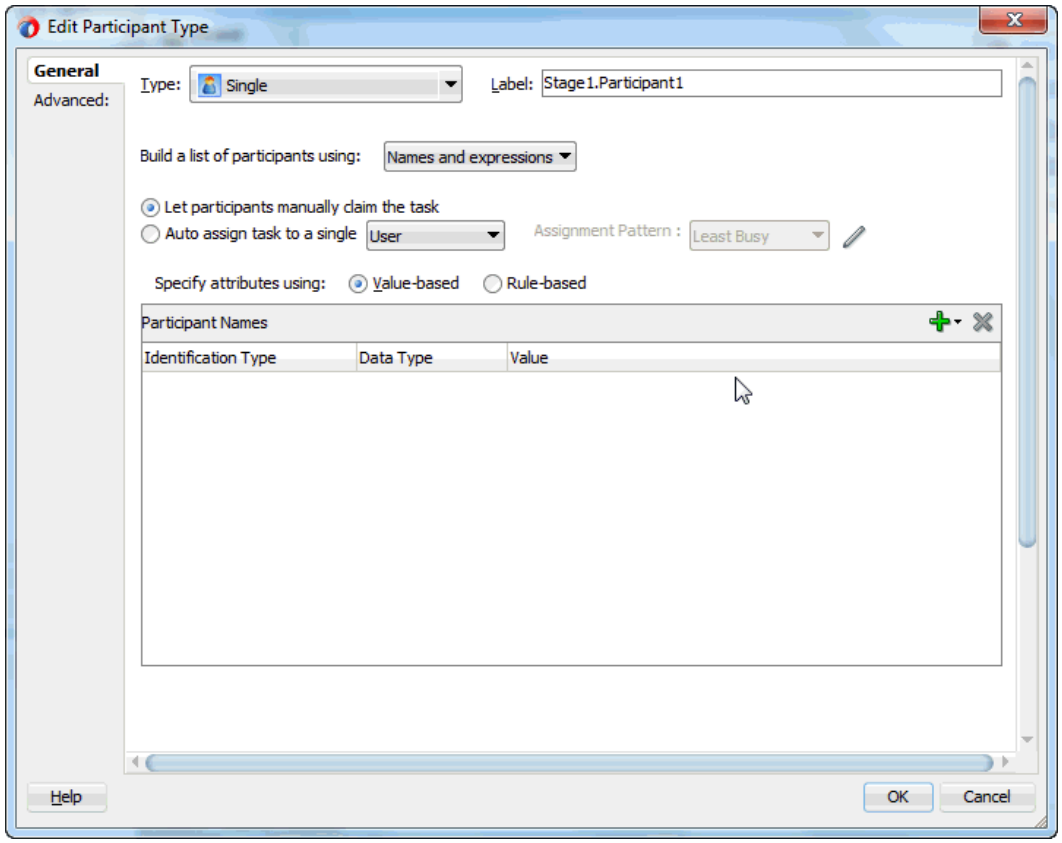
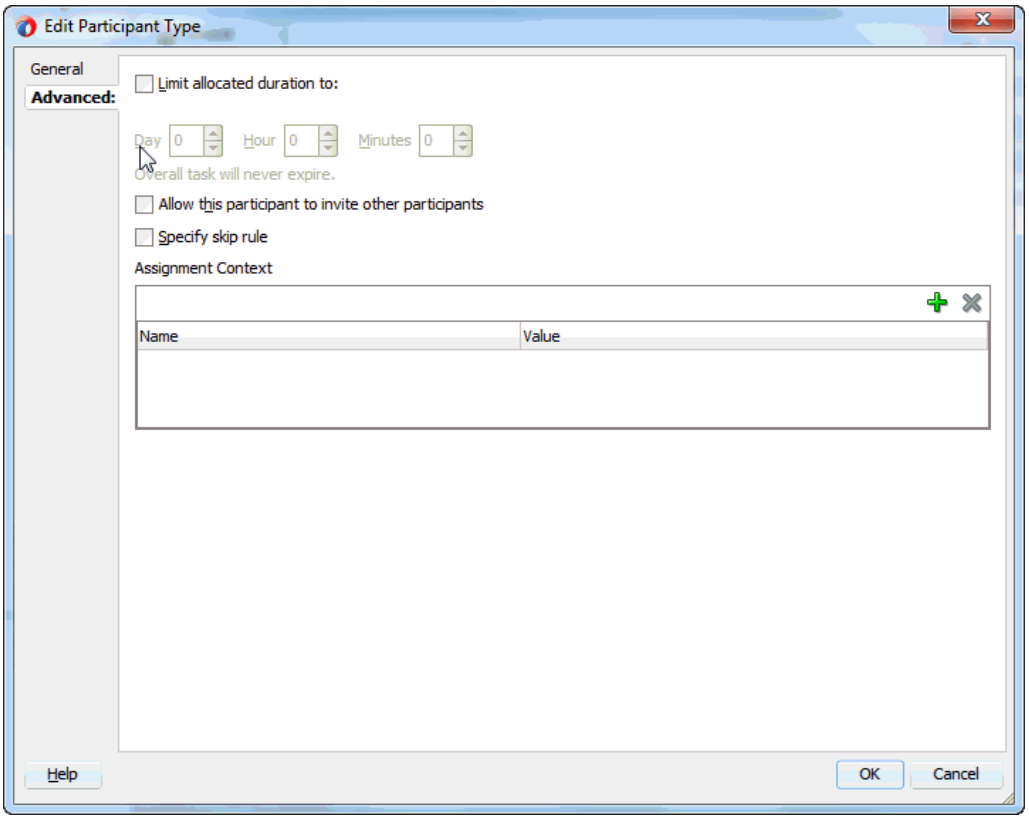


Figure 29-21 Edit Participant Type — Advanced Tab



To be dynamically assigned to a task, a single participant can be selected from a group, an application role, or a participant list.

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 box for the single participant type are listed in Table 29-7:

Table 29-7 Edit Participant Type — Single Type

For This Subsection...	See...
Participant List	Creating a Single Task Participant List
Limit allocated duration to (under the **Advanced** section)	Specifying a Time Limit for Acting on a Task
Allow this participant to invite other participants (under the **Advanced** section)	Inviting Additional Participants to a Task
Specify skip rule (under the **Advanced** section)	Bypassing a Task Participant

Table 29-7 (Cont.) Edit Participant Type — Single Type

For This Subsection...	See...
Assignment Control (under the **Advanced** section)	If this participant is associated with a particular assignment context, then add that name here. Use the Add button to add new entry. Use the drop-down list to select the assignment context Name and provide a value for this assignment context.
Let participants manually claim task (under the **General** section)	Creating Participant Lists Consisting of Value-Based Names and Expressions
Auto assign task to a single user/ group/application role (under the **General** section)	Creating Participant Lists Consisting of Value-Based Names and Expressions

Creating a Single Task Participant List

Users assigned to a participant list can act upon tasks. In a single-task participant list, only one user is required to act on the task. You can specify either a single user or a list of users, groups, or application roles for this pattern. If a list is specified, then all users on the list are assigned the task. You can specify either that one of them must manually claim and act upon the task, or that one user from the list is automatically selected by an assignment pattern. When one user acts on the task, the task is withdrawn from the task list of other assignees.

You can create several types of lists for the single user participant, and for the parallel, serial, and FYI user participants, for example:

- 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.

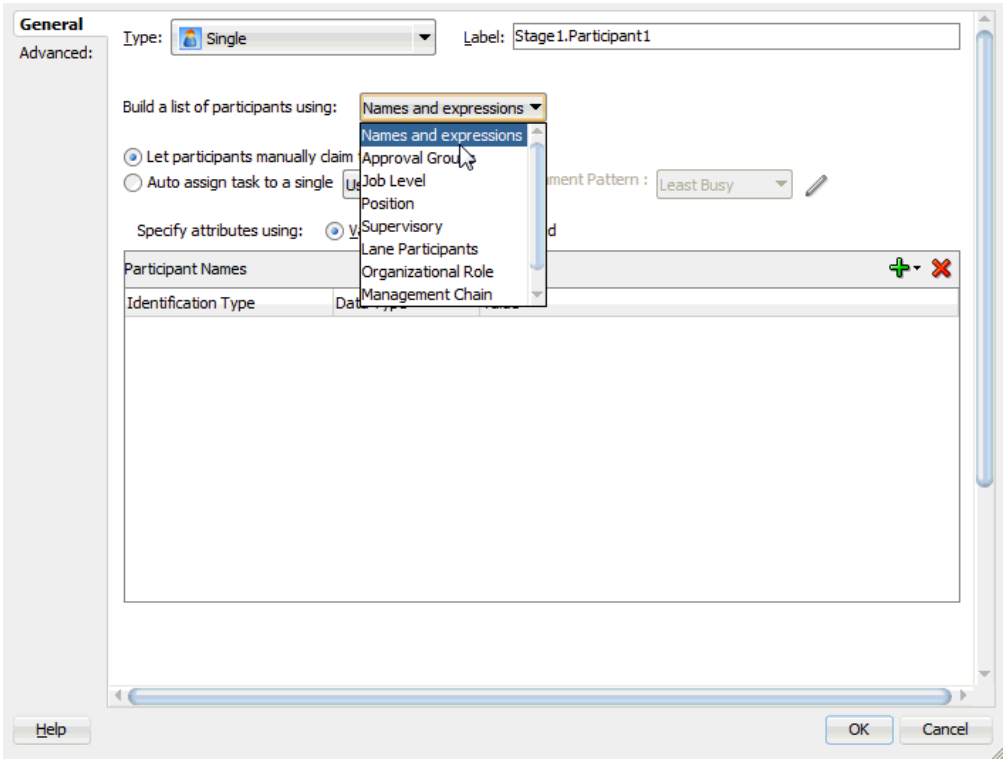
  For example, a purchase order is assigned to a manager. If the manager approves the order, it is assigned to their manager. If that manager approves it, it is assigned to their manager, and so on until three managers approve the order. If any managers reject the request or the request expires, the order 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 How to Specify Advanced Task Routing Using Business Rules.

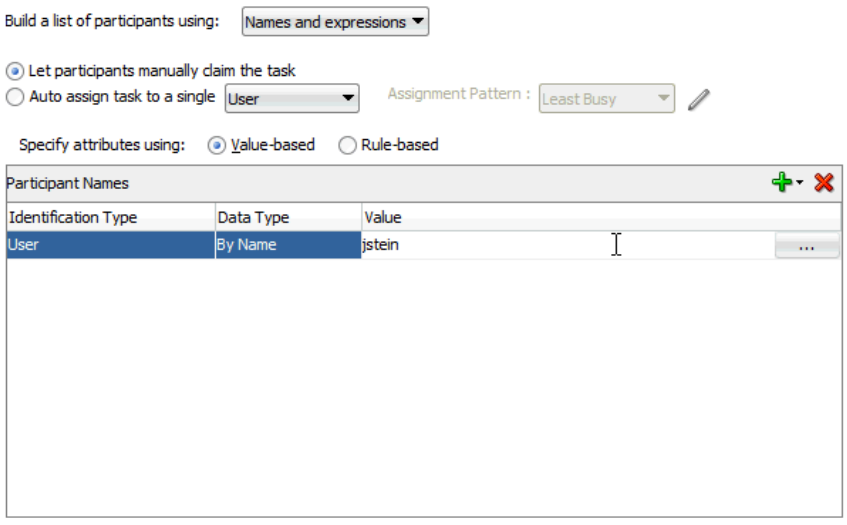
When you select a participant type, a dialog box enables you to choose an option for building your list of task participant assignees (users, groups, or application roles), as shown in Figure 29-22. The three selections described above are available: **Names and expressions**, **Management Chain**, and **Rule-based**.

Figure 29-22 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 29-23.

Figure 29-23 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. If the participant list contains a user, the selecting a group or an application role causes the dynamic assignment to fail.

For conceptual information, see the following:

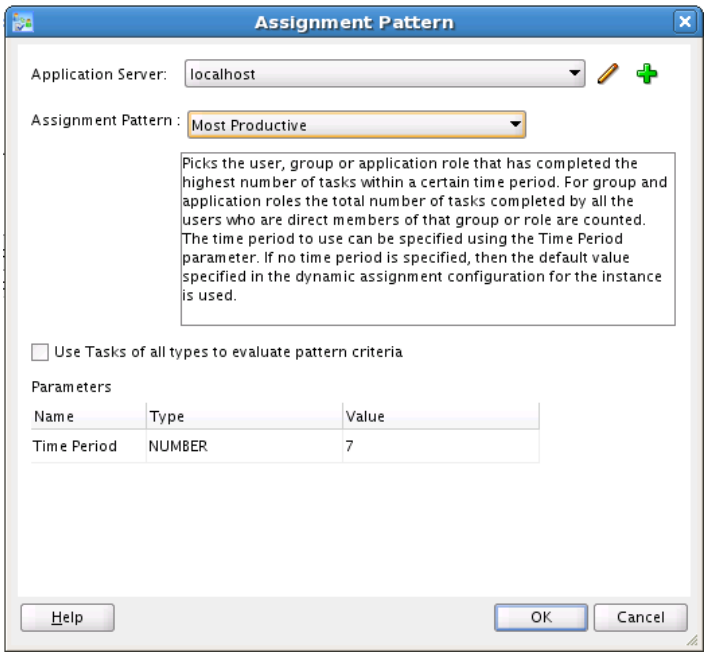
- Users, groups, or application roles, see Task Assignment and Routing.
- Statically and dynamically assigning task participants, see Static_ Dynamic_ and Rule-Based Task Assignment.

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. Do either of the following:
   - Select **Let participants manually claim the task**. If you select this option, then the task is assigned to all participants in the list. An individual user from the task assignees can then manually claim the task to work on it.
   - Select **Auto-assign to a single** list, select **User**, **Group**, or **Application Role**, then select an assignment pattern.

     To find out more about each assignment pattern, and to select and configure it, click **Assignment Pattern**. The Assignment Pattern dialog box appears. Figure 29-24 shows an example of an Assignment Pattern dialog box.

Figure 29-24 Selecting and Configuring an Assignment Pattern



When you specify an application server connection in the Application Server field, the assignment patterns are loaded into the Assignment Pattern list. When you select one of the patterns from the Assignment Pattern list, a description of your selection appears in the text box.

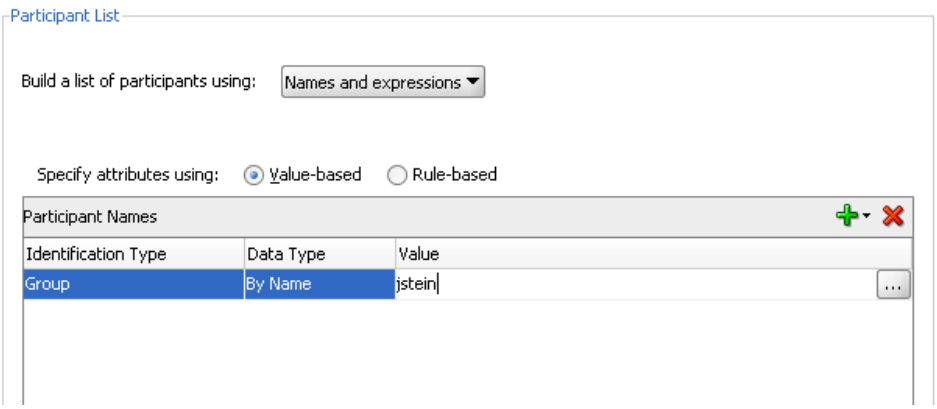
If you want the assignment pattern to consider all types of tasks, then select **Use tasks of all types to evaluate pattern criteria**. Otherwise, the pattern considers only this task type when determining the selected user. For example, to assign a vacation request task to the least busy user, and you select **Use tasks of all types to evaluate pattern criteria**, then all assigned tasks are taken into consideration when determining the least busy user. If you do *not* select **Use tasks of all types to evaluate pattern criteria**, then only assigned vacation request tasks are considered when determining the least busy user.

A particular pattern may enable you to specify input parameters that control how the pattern is evaluated. For example, as shown in Figure 29-24, the Most Productive pattern enables you to specify the Time Period (in days) over which the productivity is calculated. Input values can be static, or can be dynamically set by using an XPath expression. Not all patterns accept parameters.

3. From the **Specify attributes using** list, select **Value-based**.

   The dialog box refreshes to display the fields shown in Figure 29-25.

Figure 29-25 Value-Based Names and Expressions



4. 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.

5. To change your selection in the **Identification Type** column, click it to invoke a drop-down list.

6. In the **Data Type** column, click your selection to invoke a drop-down 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 box 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 box 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. 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 box. Use the `bpws:getVariableData(...)` expression or the `ids:getManager()` XPath function.

The **Value** column displays the value you specified.

7. 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 Task Assignment and Routing.
- Statically and dynamically assigning task participants, see Static_ Dynamic_ and Rule-Based Task Assignment.
- Management chains, see Creating a Single Task Participant List.

To create participant lists based on value-based management chains:

1. From the **Build a list of participants using** list, select **Management Chain**.
2. Do either of the following:
   - Select **Let participants manually claim the task**. If you select this option, then the task is assigned to all participants in the list. An individual user from the task assignees can then manually claim the task to work on it.
   - Select **Auto-assign to a single** list, select **User**, then select an assignment pattern.

     To find out more about each assignment pattern, and to select and configure it, click **Assignment Pattern**. The Assignment Pattern dialog box appears. Figure 29-24 shows an example of an Assignment Pattern dialog box.

     When you specify an application server connection in the Application Server field, the assignment patterns are loaded into the Assignment Pattern list. When you select one of the patterns from the Assignment Pattern list, a description of your selection appears in the text box.

     If you want the assignment pattern to consider all types of tasks, then select **Use tasks of all types to evaluate pattern criteria**. Otherwise, the pattern considers only this task type when determining the selected user. For example, to assign a vacation request task to the least busy user, and you select **Use tasks of all types to evaluate pattern criteria**, then all assigned tasks are taken into consideration when determining the least busy user. If you do *not* select **Use tasks of all types to evaluate pattern criteria**, then only assigned vacation request tasks are considered when determining the least busy user.

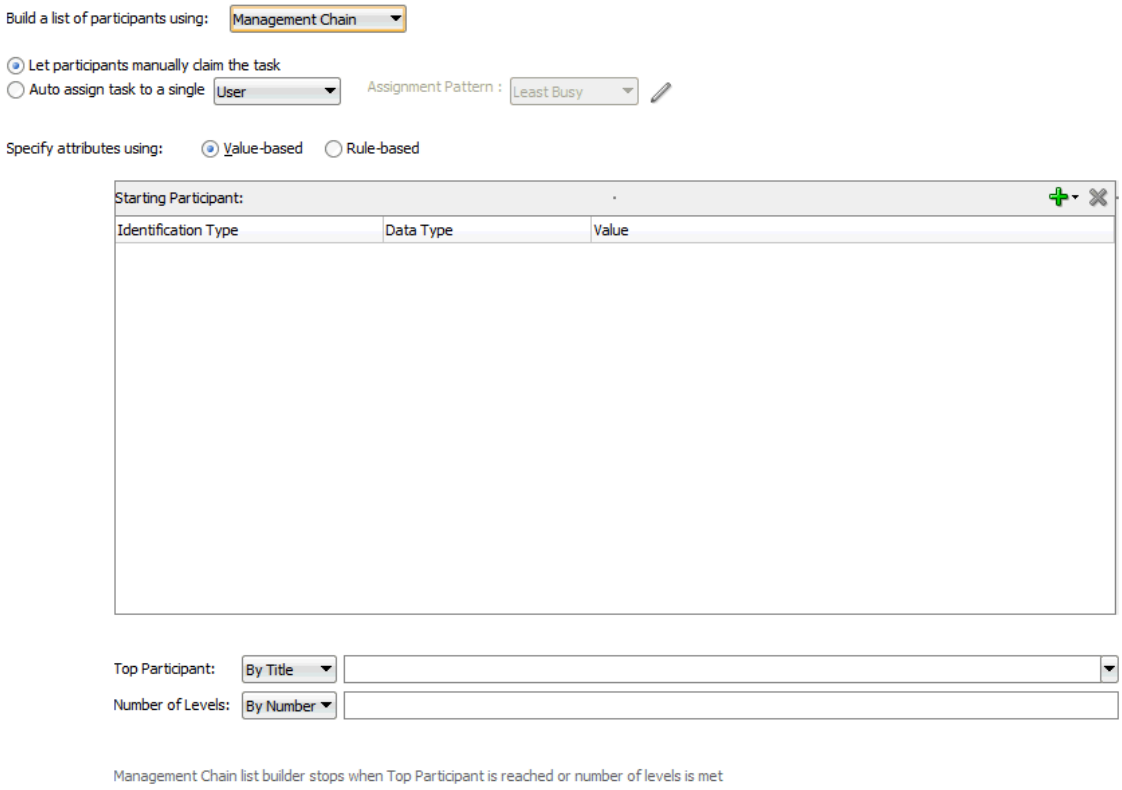
     A particular pattern may enable you to specify input parameters that control how the pattern is evaluated. For example, as shown in Figure 29-24, the Most Productive pattern enables you to specify the Time Period (in days) over which

the productivity is calculated. Input values can be static, or can be dynamically set by using an XPath expression. Not all patterns accept parameters.

3. From the **Specify attributes using** list, select **Value-based**.

   The dialog box refreshes to display the fields shown in Figure 29-26.

Figure 29-26 Value-Based Management Chains



4. See Step 4 through Step 7 of Creating a Single Task Participant List for instructions on assigning a user, group, or application role to a list in the **Starting Participant** table.

5. 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 box.

6. 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 box.

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.

Note:

You cannot update facts after the rule dictionary is created.

To specify participant lists based on rulesets:

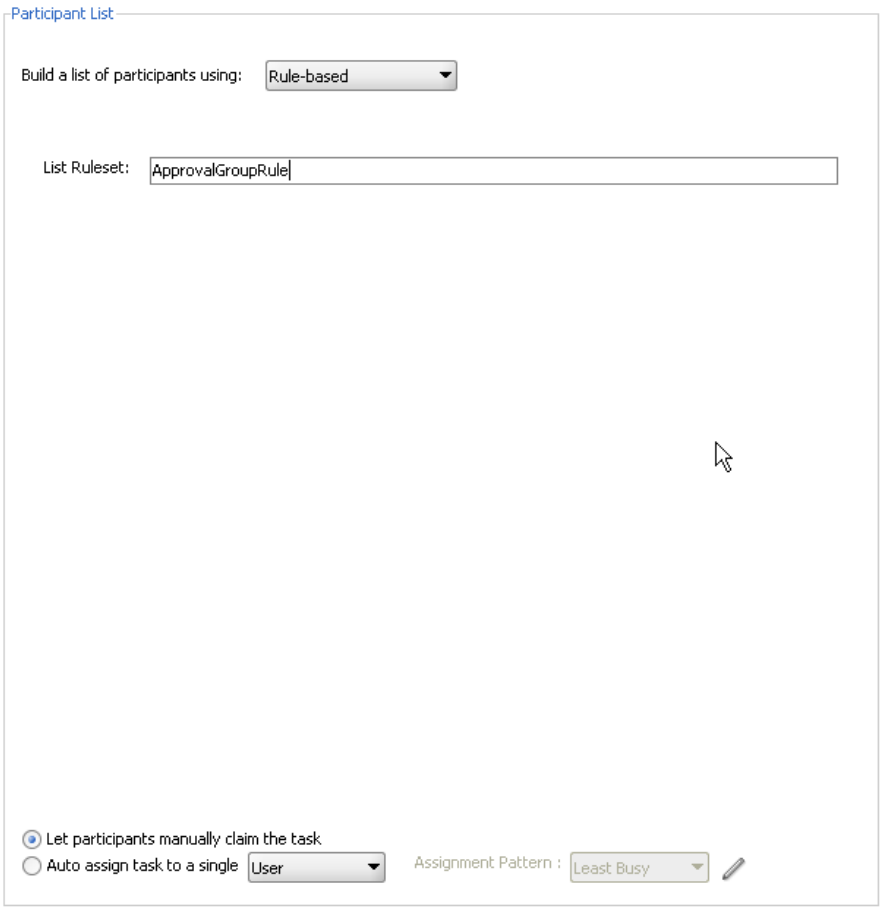
Business rules can define the participant list. There are two options for using business rules:

- 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 29-27 provides details.

Figure 29-27 Rulesets



4. Do either of the following:

   – Select **Let participants manually claim the task**. If you select this option, then the task is assigned to all participants in the list. An individual user from the task assignees can then manually claim the task to work on it.

   – Select **Auto-assign to a single** list, select **User**, **Group**, or **Application Role**, then select an assignment pattern.

     To find out more about each assignment pattern, and to select and configure it, click **Assignment Pattern**. The Assignment Pattern dialog box appears. Figure 29-24 shows an example of an Assignment Pattern dialog box.

     When you specify an application server connection in the Application Server field, the assignment patterns are loaded into the Assignment Pattern list. When you select one of the patterns from the Assignment Pattern list, a description of your selection appears in the text box.

     If you want the assignment pattern to consider all types of tasks, then select **Use tasks of all types to evaluate pattern criteria**. Otherwise, the pattern considers only this task type when determining the selected user. For example, to assign a vacation request task to the least busy user, and you select **Use tasks of all types to evaluate pattern criteria**, then all assigned tasks are taken into consideration when determining the least busy user. If you do *not* select **Use tasks of all types to evaluate pattern criteria**, then

only assigned vacation request tasks are considered when determining the least busy user.

A particular pattern may enable you to specify input parameters that control how the pattern is evaluated. For example, as shown in Figure 29-24, the Most Productive pattern enables you to specify the Time Period (in days) over which the productivity is calculated. Input values can be static, or can be dynamically set by using an XPath expression. Not all patterns accept parameters.

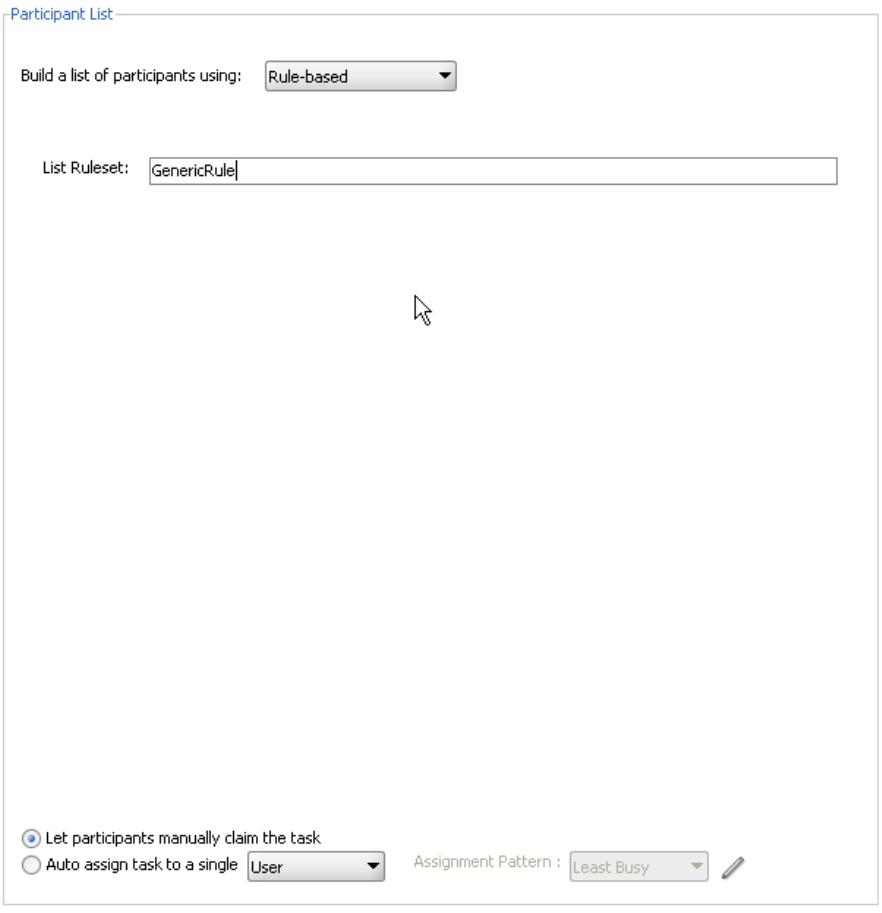
5. Click **OK**.

- 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 29-28 provides details.

Figure 29-28 Rulesets



3. Do either of the following:

   – Select **Let participants manually claim the task**. If you select this option, then the task is assigned to all participants in the list. An individual user from the task assignees can then manually claim the task to work on it.

– Select **Auto-assign to a single** list, select **User**, **Group**, or **Application Role**, then select an assignment pattern.

  To find out more about each assignment pattern, and to select and configure it, click **Assignment Pattern**. The Assignment Pattern dialog box appears. Figure 29-24 shows an example of an Assignment Pattern dialog box.

  When you specify an application server connection in the Application Server field, the assignment patterns are loaded into the Assignment Pattern list. When you select one of the patterns from the Assignment Pattern list, a description of your selection appears in the text box.

  If you want the assignment pattern to consider all types of tasks, then select **Use tasks of all types to evaluate pattern criteria**. Otherwise, the pattern considers only this task type when determining the selected user. For example, to assign a vacation request task to the least busy user, and you select **Use tasks of all types to evaluate pattern criteria**, then all assigned tasks are taken into consideration when determining the least busy user. If you do *not* select **Use tasks of all types to evaluate pattern criteria**, then only assigned vacation request tasks are considered when determining the least busy user.

4. 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`

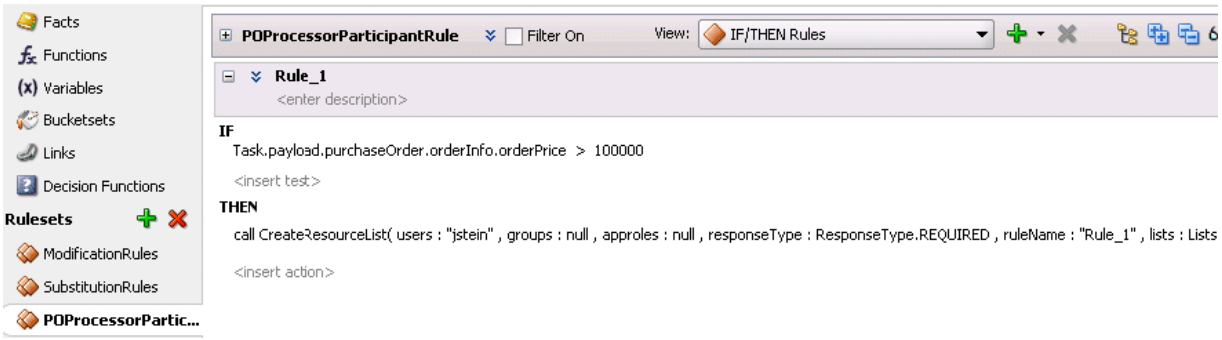
The `Task` fact is asserted by the task service for basing rule conditions.

Viewing the Rule Dictionary

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 29-29 provides details.

Figure 29-29 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 29-30.

Figure 29-30 Business Rules

If multiple rules are fired, the list builder created by the rule with the highest priority is selected.

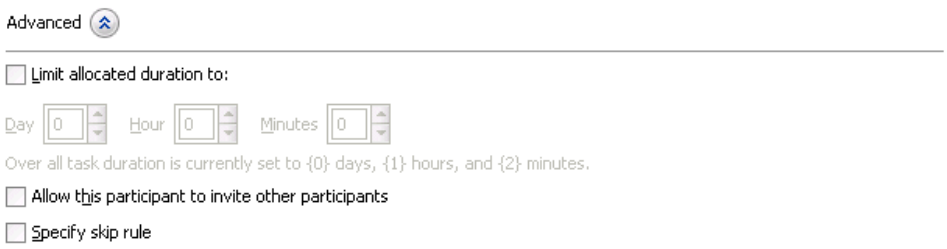
Specifying a Time Limit for Acting on a Task

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 box for the single type, as shown in Figure 29-31.

   Figure 29-31 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 Escalating_ Renewing_ or Ending the Task.

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.

Note:

Do not add adhoc assignees either above or below an FYI participant.

To invite additional participants to a task:

1. Expand the **Advanced** section of the Edit Participant Type dialog box for the single type, as shown in Figure 29-31.
2. Select **Allow this participant to invite other participants**.

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 box for the single type, as shown in Figure 29-31.
2. Select **Specify skip rule**.

   This action displays an icon for accessing the Expression Builder dialog box 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 How to Specify Advanced Task Routing Using Business Rules.

How to Configure the Parallel Participant Type

The parallel 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. In case of parallel routing with parallel participants, the voting and the percentage rule takes precedence to decide the final outcome of the parent task.

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 29-32 and Figure 29-33 display the upper and lower sections of the Parallel dialog box.

Figure 29-32 Edit Participant Type — Parallel Type (Upper Section of Dialog)

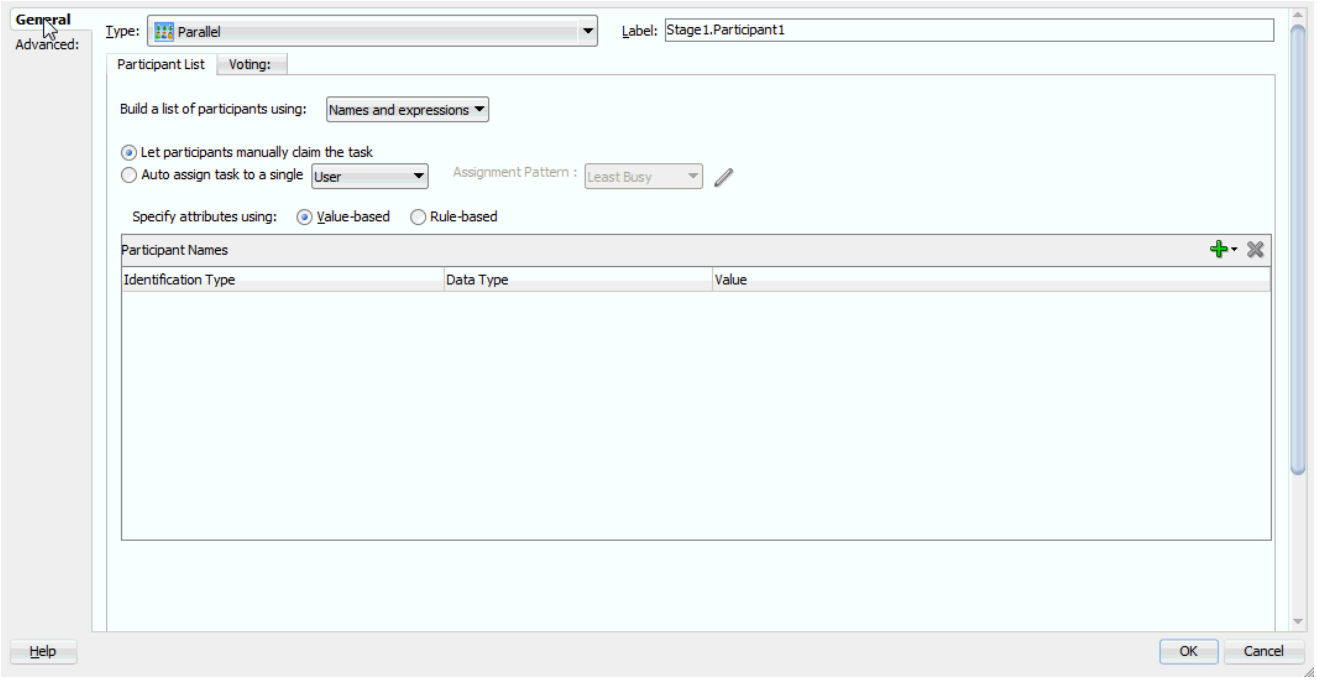
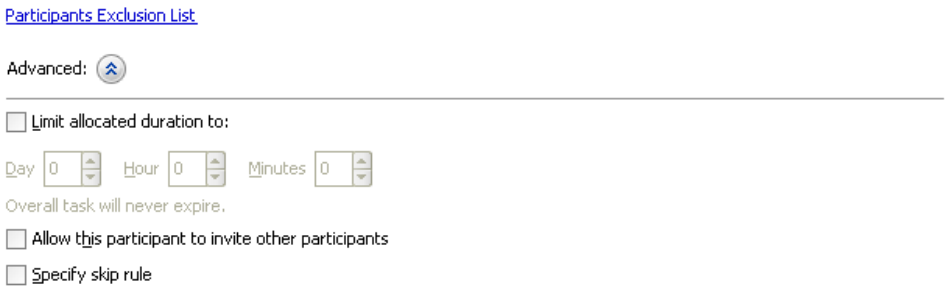


Figure 29-33 Edit Participant Type — Parallel Type (Lower Section of Dialog)



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 box for the parallel participant type are listed in Table 29-8:

Table 29-8 Edit Participant Type — Parallel Type

For This Subsection...	See...
Vote Outcome	Specifying the Voting Outcome
Participant List	Creating a Parallel Task Participant List

Table 29-8 (Cont.) Edit Participant Type — Parallel Type

For This Subsection...	See...
Limit allocated duration to (under the **Advanced** section)	Specifying a Time Limit for Acting on a Task
Allow this participant to invite other participants (under the **Advanced** section)	Inviting Additional Participants to a Task
Specify skip rule (under the **Advanced** section)	Bypassing a Task Participant
Add Assignment Context (under the **Advanced** section)	If this participant is associated with a particular assignment context, then add that name here. Use the Add button to add a new entry. Use the drop-down list to select an assignment context Name and to provide a value for this assignment context.

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.

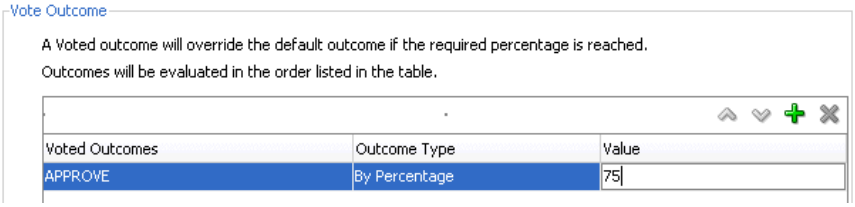
To specify group voting details:

1. Go to the **Vote Outcome** section of the Edit Participant Type dialog box 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 How to Specify 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 box 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 29-34 provides details.

This functionality is nondeterministic. For example, selecting a percentage of **30%** when there are two subtasks does not make sense.

Figure 29-34 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 box in How to Specify a Task Outcome display in this list.

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 Creating a Single Task Participant List.

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 box for the parallel type, click the **Advanced** tab to expand the section shown in Figure 29-33.
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 Escalating_ Renewing_ or Ending the Task.

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 box for the parallel type, click the **Advanced** icon to expand the section (if not expanded).
2. Select **Allow this participant to invite other participants**.

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 box for the parallel type, select the **Specify skip rule** check box.

   This action displays an icon for accessing the Expression Builder dialog box for building a condition. The expression must evaluate to a boolean value.

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

How to Configure the Serial Participant Type

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 29-35 displays the Serial dialog box. Figure 29-36 shows the expanded **Advanced** section.

Figure 29-35 Edit Participant Type — Serial Type

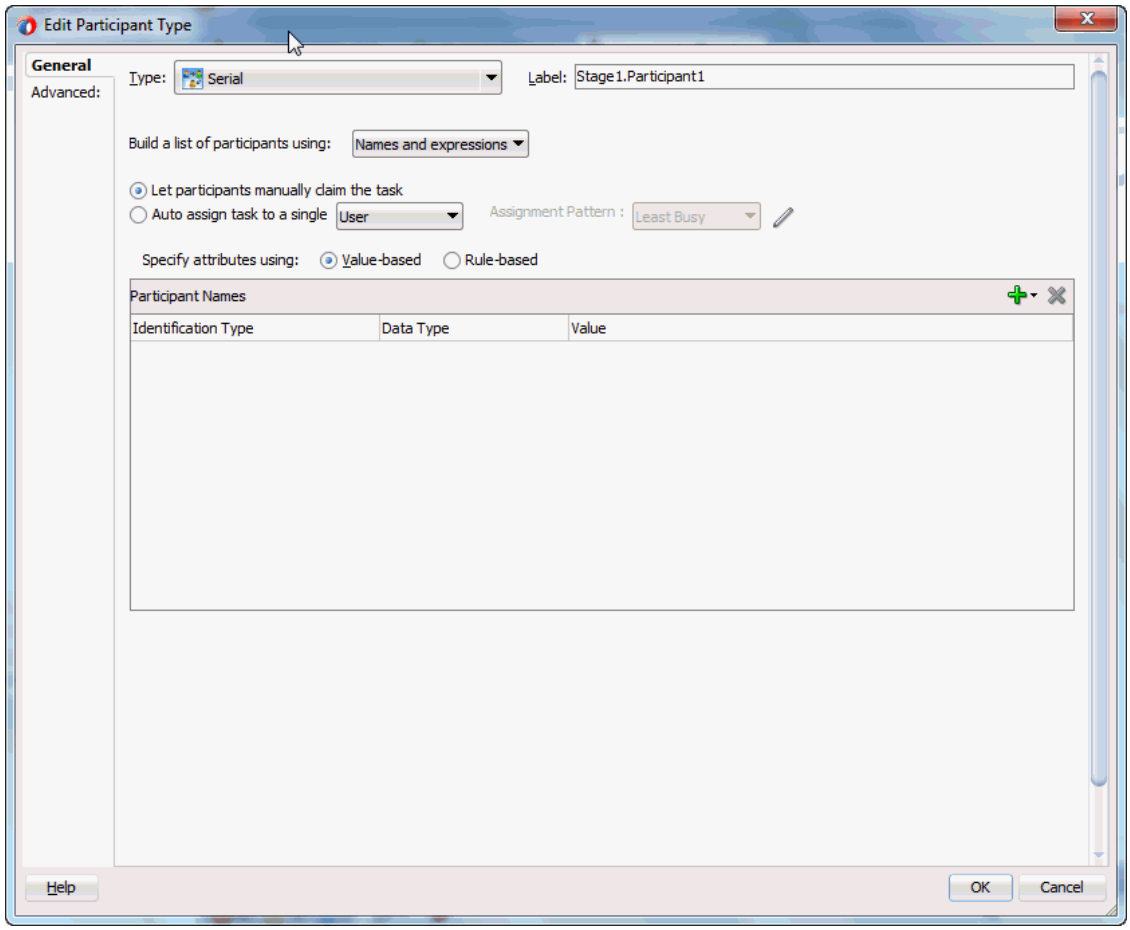
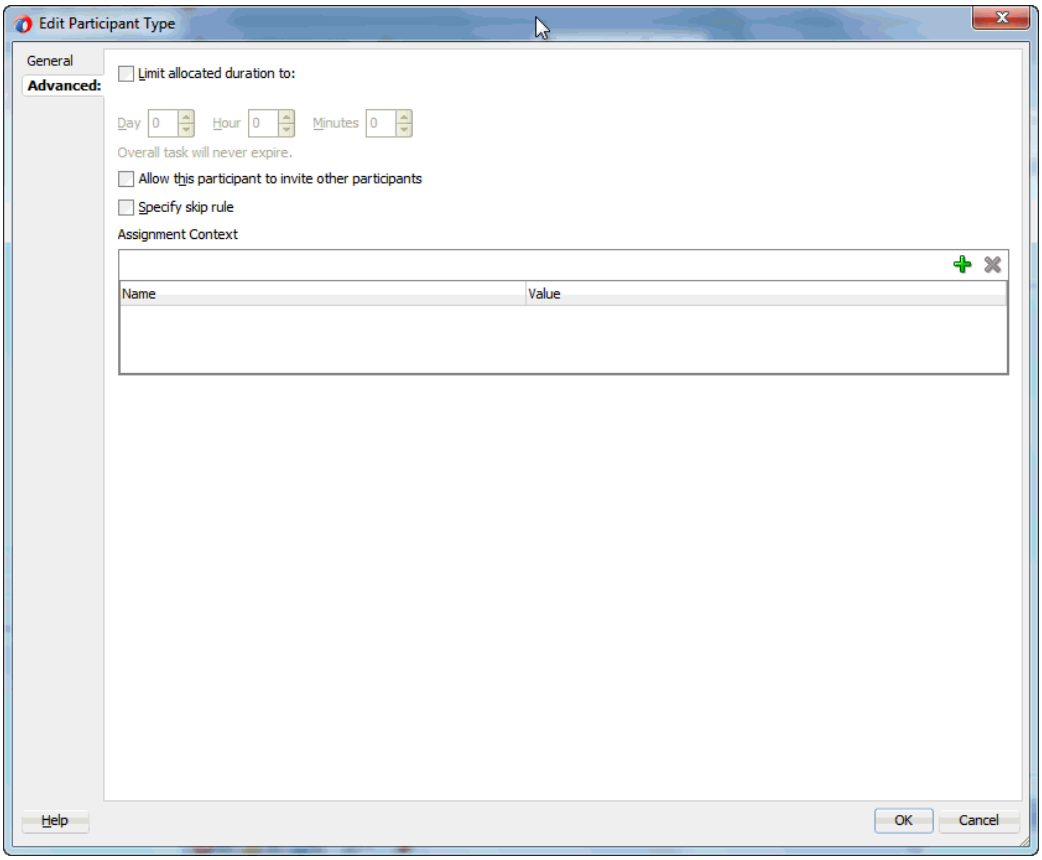


Figure 29-36 Edit Participant Type — Serial Type (Advanced Tab)



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 box for the serial participant type are listed in Table 29-9.

Table 29-9 Edit Participant Type — Serial Type

For This Subsection...	See...
Participant List	Creating a Serial Task Participant List
Limit allocated duration to (under the **Advanced** section)	Specifying a Time Limit for Acting on a Task Note that if you specify the task expiry time at the level of a serial participant, then, when that time expires, the task does not move to the next participant in the series. Rather, the entire task expires.
Allow this participant to invite other participants (under the **Advanced** section)	Inviting Additional Participants to a Task

Table 29-9 (Cont.) Edit Participant Type — Serial Type

For This Subsection...	See...
Specify skip rule (under the **Advanced** section)	Bypassing a Task Participant
Assignment Context (under the **Advanced** section)	If this participant is associated with a particular assignment context, then add that name here. Use the Add button to add a new entry. Use the drop-down list to select assignment context Name and to provide a value for this assignment context.

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 Creating a Single Task Participant List for instructions on creating these lists of participants.

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** tab of the Edit Participant Type dialog box for the serial type, click the **Advanced** icon to expand the section shown in Figure 29-35.
2. Click **Limit allocated duration to**.
3. Specify the amount of time.

Note:

If you specify the task expiry time at the level of a serial participant, then, when that specified time limit is reached, the task does not move to the next participant in the series. Rather, the entire task expires.

For more information about setting the global escalation and renewal policies in the **Deadlines** section of the Human Task Editor, see Escalating_ Renewing_ or Ending the Task.

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 box 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 series is complete.

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.

In the **Advanced** section of the Edit Participant Type dialog box for the serial type, select the **Specify skip rule** check box to bypass a task participant. This action displays an icon for accessing the Expression Builder dialog box 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 Bypassing a Task Participant.

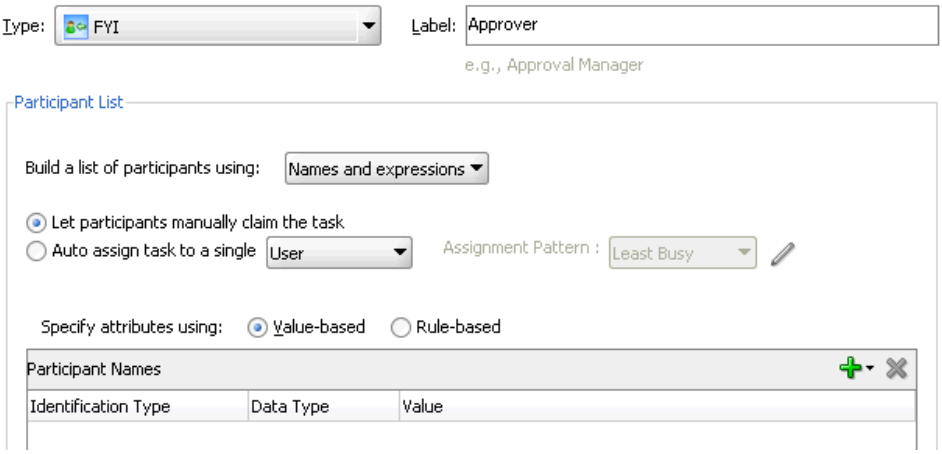
How to Configure the FYI Participant Type

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 29-37 displays the Edit Participant Type dialog box for the FYI type. This dialog box also includes a **Participants Exclusion List** at the bottom that is not displayed in Figure 29-37.

Figure 29-37 Edit Participant Type — FYI Type



To configure the FYI participant type, 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).

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 Creating a Single Task Participant List for instructions on creating these lists of participants.

Selecting a Routing Policy

After you configure a participant type and are returned to the Human Task Editor, use the links on the top right corner as shown in Figure 29-38.

Figure 29-38 Human Task Editor — Assignment Section

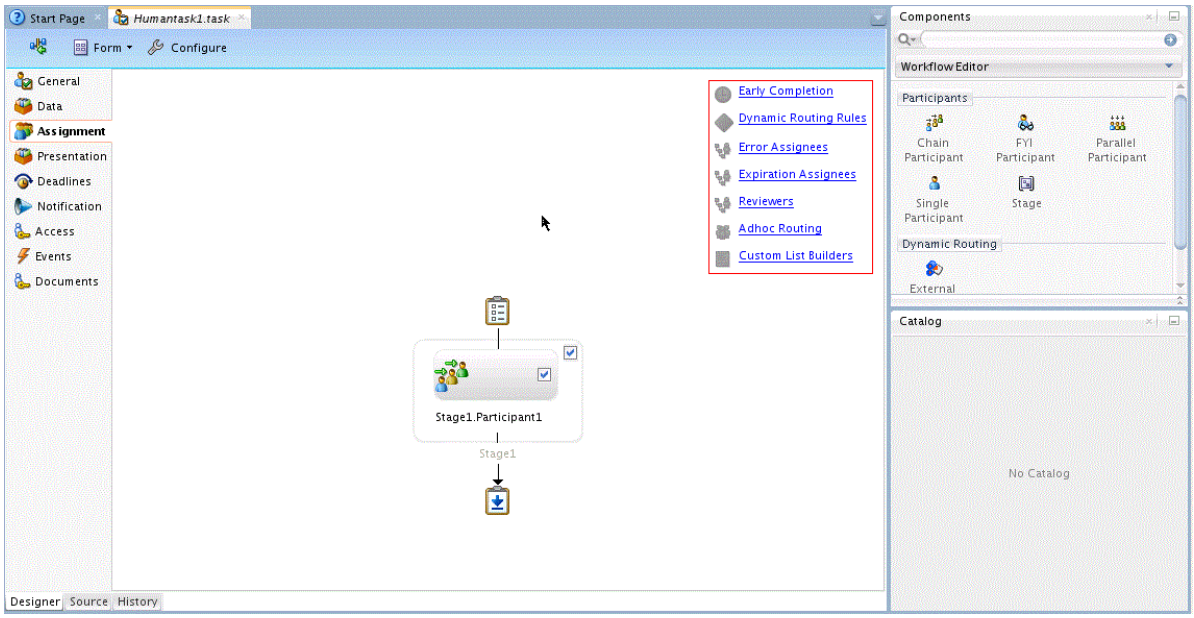


Table 29-10 describes the routing policy methods provided.

Table 29-10 Routing Policy Method

Routing Policy Selection	Use This Policy In Environments Where...	Section
• **Allow all participants to invite other participants**	A participant can select users or groups as the next assignee (ad hoc) when approving the task.	Allow All Participants to Invite Other Participants or Edit New 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.	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.	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.	Completing Parent Subtasks of Early Completing Subtasks

Table 29-10 (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.	How to Specify 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.	How to Use External Routing
Assignment tab	A participant is assigned a failed task for the purposes of recovery.	How to Configure the Error Assignee and Reviewers

How to Customize Tasks Routing

Tasks are reviewed by all the selected participants in the order they appear. This is the default routing. However, you can add some Adhoc or Dynamic routing rules.

Dynamic and Adhoc Routing Rules

Dynamic and Adhoc Routing help you with the following:

- 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

Allow All Participants to Invite Other Participants or Edit New Participants

This check box 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. Click **Adhoc Routing**.
2. Select the **Allow all participants to invite other participants** check box for this task assignee to invite other participants into the workflow before routing it to the next assignee in this workflow.
3. Select the **Allow participants to edit new participants** check box for this task assignee to edit other adhoc participants that were added to the routing slip.

Note:

Do not add adhoc assignees either above or below an FYI participant.

Allow Initiator to Add Participants

In the **Adhoc Routing** screen, select the **Allow all initiator to add participants** check box so this task initiator can invite other participants into the workflow before routing to the next assignee in this workflow.

Stopping Routing of a Task to Further Participants

You can specify conditions under which a task can be marked complete 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. Click **Early Completion**.
2. Select the **Complete task when a participant chooses: <outcome>** check box.

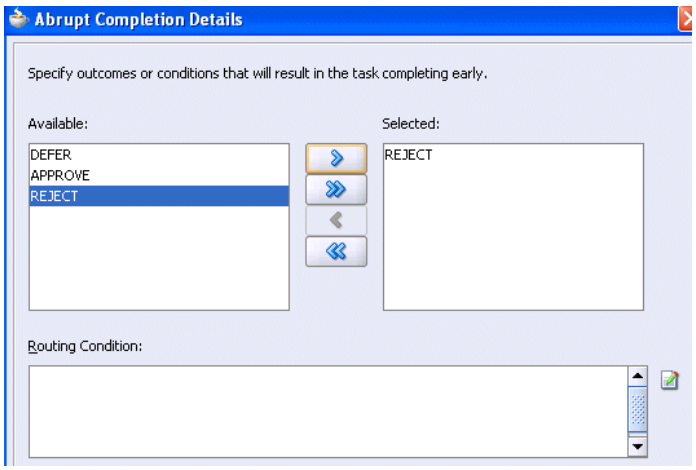
   The **Abrupt Completion Details** screen appears. There are two methods for specifying the abrupt completion of a task:

   - 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.

3. Select appropriate outcomes and click the > button, as shown in Figure 29-39. To select all, click the >> button.

Figure 29-39 Abrupt Completion Details



4. To the right of the **Routing Condition** field, click the icon to display the Expression Builder dialog box 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.

An early completion XPath expression is not evaluated until at least one user has acted upon the task.

5. To enable early completion, click **Enable early completion in parallel with subtasks**. For more information, see Enabling Early Completion in Parallel Subtasks.

6. To enable early completion of parent tasks, click **Complete parent tasks of early completing subtasks**. For more information, see Completing Parent Subtasks of Early Completing Subtasks.

7. 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>** check box to edit this information.

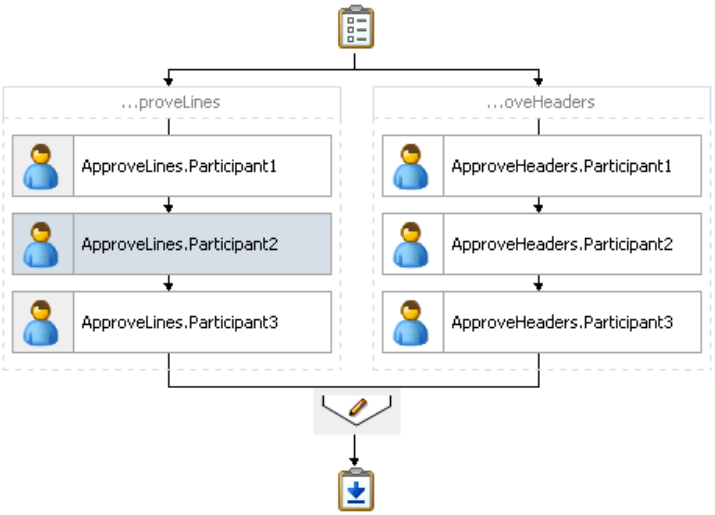
Enabling Early Completion in Parallel Subtasks

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 29-40 provides details.

Figure 29-40 Early Completion of Parallel Subtasks



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 29-40. 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.

How to Specify 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.

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 How to Route 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.

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 29-11 describes the fact types asserted by the task service.

Table 29-11 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 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 29-12 describes where you can use each type.

Table 29-12 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
`ResourceListType`	No	Yes
`ManagementChainListType`	No	Yes
`ResourceType`	No	Yes
`ParameterType`	No	Yes
`AutoActionType`	No	Yes

Table 29-12 (Cont.) Use of Fact Types

Fact Type	Can Use in Routing Rules?	Can Use in Participant Rules?
`ResponseType`	No	Yes

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 29-13:

Table 29-13 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). **Note:** Pushback is designed to work with single approvers and not with group votes. Pushback from a stage with group vote (or parallel) scenario to another stage is not allowed. Similarly, you cannot push back from a single assignee to a group vote (or parallel) scenario.	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

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:

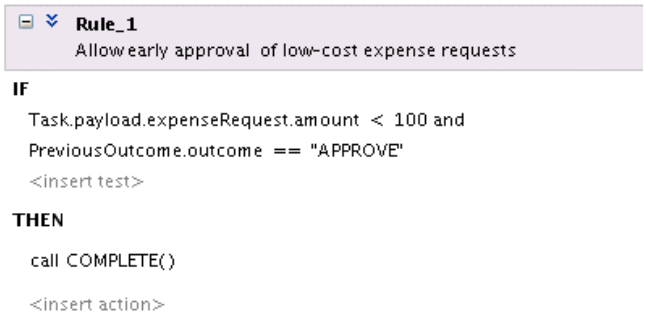
- If the total amount of the expense request is less than $100, approval is only required from one of the participants. Otherwise, it must be approved by all three.
- If an expense request is rejected by any of the participants, it must be returned to the previous participant for re-evaluation. If it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

This behavior is implemented using the following rules. 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:

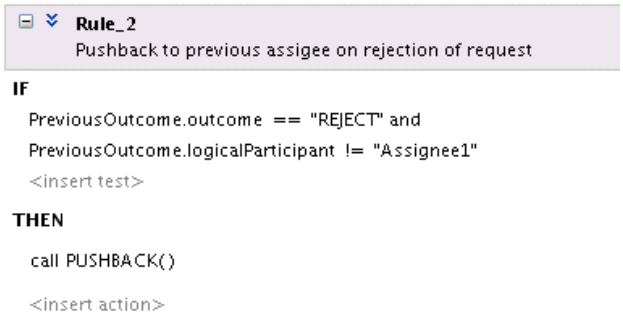
- Early approval rule (Figure 29-41):

Figure 29-41 Early Approval Rule



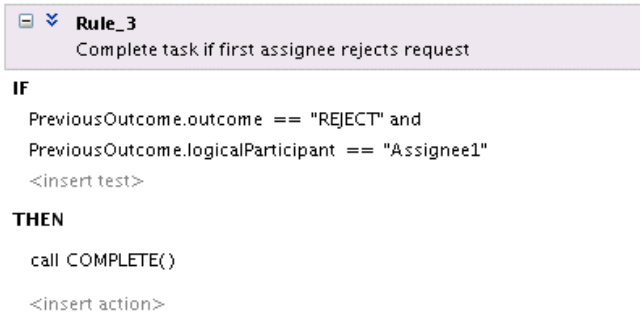
- Push back on the rejected rule (Figure 29-42):

Figure 29-42 Push Back On The Rejected Rule



- Complete the `Assignee1` rejected rule (Figure 29-43):

Figure 29-43 Completion of the Assignee1 Rejected Rule



For information about iterative design, see the `workflow-106-IterativeDesign` sample available with the Oracle SOA Suite samples.

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 Customizing SOA Composite Applications .

Creating Advanced Routing Rules

To create advanced routing rules:

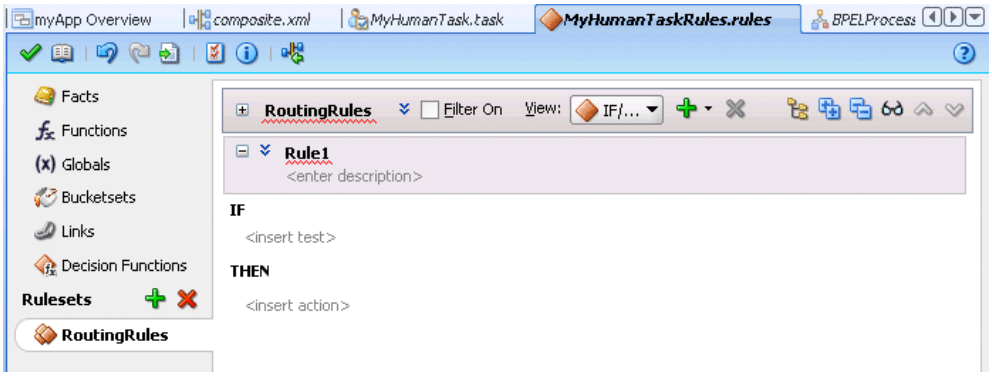
1. In the **Assignment** section, click **Dynamic Routing Rules**.

   The Use Advanced Rules edit box displays.

2. Click **Create Rules**.

   This starts the Oracle Business Rules Designer with a pre-seeded repository containing all necessary fact definitions, as shown in Figure 29-44. A decision service component is created for the dictionary, and is associated with the task service component.

Figure 29-44 Human Task Rule Dictionary



3. Define state machine routing rules for your task using Oracle Business Rules.

   This automatically creates a fully-wired decision service in the human task and the associated rule repository and data model.

   To edit the business rules, click the **Edit** icon, next to the **Rules Dictionary** field.

   For more information about business rules, see the following documentation:

   - Sample Ruleset for an example human task ruleset
   - *Designing Business Rules with Oracle Business Process Management*
   - *Rules Language Reference for Oracle Business Process Management*

How to Use 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 box, specify a Java class, and click **OK** to exit, the next time you open this dialog box, the other two selections (**Route task to all participants, in order specified** and **Use Advanced Rules**) no longer appear in the drop-down list. To access all three selections again, you must delete the entire assignment.

To use external routing

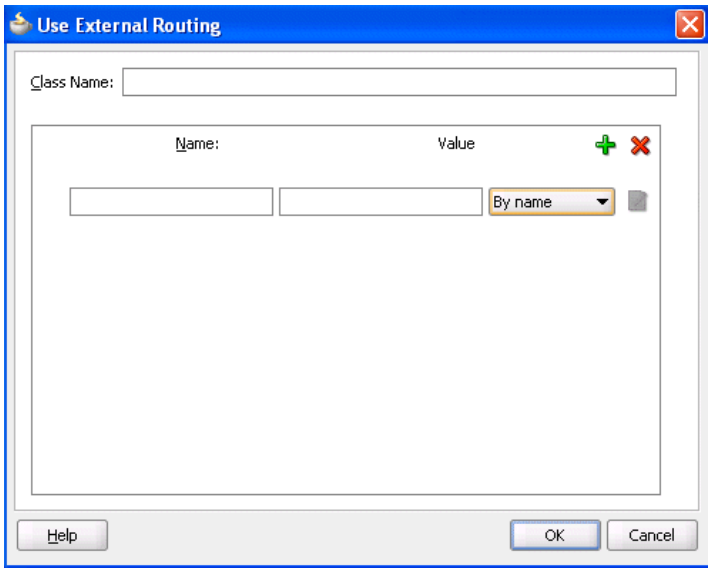
1. Drag and drop **External Routing Service** from the Workflow Editor Components window.

   The Use External Routing edit box displays.

2. Click the **Edit** icon.

The External Routing dialog box appears, as shown in Figure 29-45.

Figure 29-45 Use External Routing Dialog



3. 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`

4. Add name and pair value parameters by name or XPath expression that can be passed to the external service, as shown in Table 29-14.

Table 29-14 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 box.

5. Click the **Add** icon to add additional name and pair value parameters.

How to Configure the Error Assignee and Reviewers

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. **Note:** Do not add adhoc assignees either above or below a FYI participant.

- Reassign

  Reassign the task to the actual users assigned to this task

- Error task

  Indicate that this task cannot be rectified.

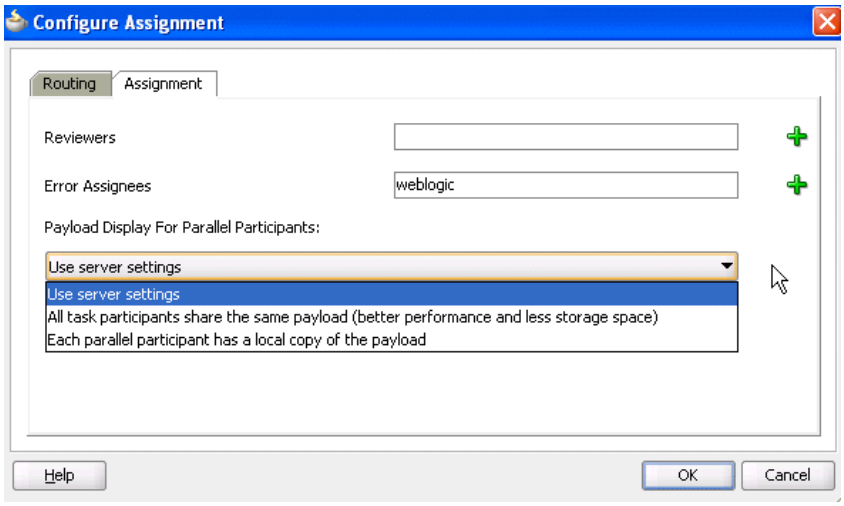
If there are any errors in evaluating the error assignees, the task is marked as being in error.

This dialog box 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. Click the **Add** icon to assign reviewers or error assignees, as shown in Figure 29-46.

Figure 29-46 Error Assignment Details



2. 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.

3. See Step 5 through 7 of Creating a Single Task Participant List for instructions on selecting a user, group, or application role.

4. 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 the settings, see How to Change Server Settings.

   - **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 situation 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.

5. Click **OK**.

For more information about users, groups, or application roles, see Task Assignment and Routing.

How to Change Server Settings

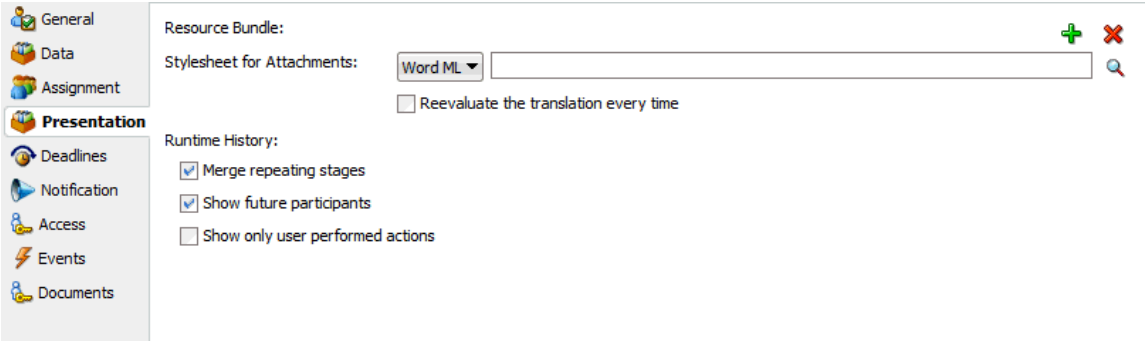
To change the default setting of **SharePayloadAcrossAllParallelApprovers** property, perform the following steps:

1. Right-click **soa-infra** and select **Administration** > **System MBean Browser**.
2. Expand **Application Defined MBeans** > **oracle.as.soainfra.config** > **Server:** *server_name* > **WorkflowConfig** > **human-workflow**.
3. Click **SharePayloadAcrossAllParallelApprovers**.
4. Change this property in the list, and click **Apply**.

Specifying Multilingual Settings and Style Sheets

The **Presentation** section shown in Figure 29-47 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 29-47 Presentation Section



How to Specify 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.

How to Specify 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:

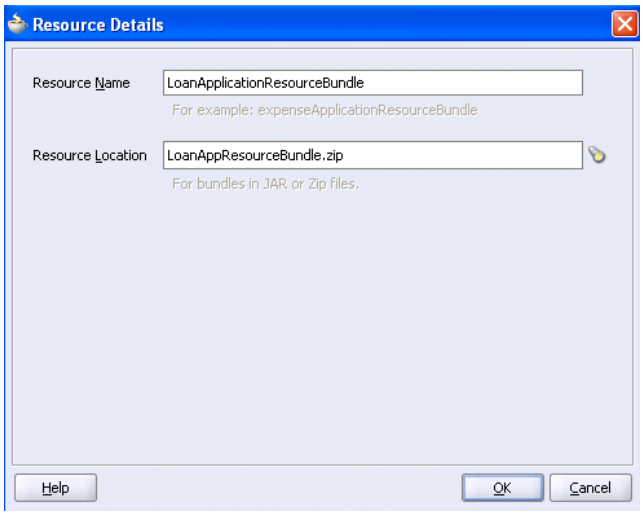
- `APPROVE=Approve`
- `REJECT=Reject`

To specify multilingual settings:

1. In the **Presentation** section, click the **Add** icon across from **Resource Bundle**.

   The Resource Details dialog box shown in Figure 29-48 appears.

   Figure 29-48 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 How to Configure Notification Messages in Different Languages.

Specify What to Show in Task Details in the Worklist

The Presentation section enables you to specify the records in the runtime history section of the task details form in `worklistapp`.

Merge repeating stages: Select this option to view one aggregated entry for all repeating stages. The Worklist UI also provides an option to set or unset this option.

Show future participants: Select this option to see details about all future participants in the task.

Show only user performed actions: By default, task history details contain records for Admin and system actions, such as root task updates. Select this option to not see only user-performed action updates in the task details.

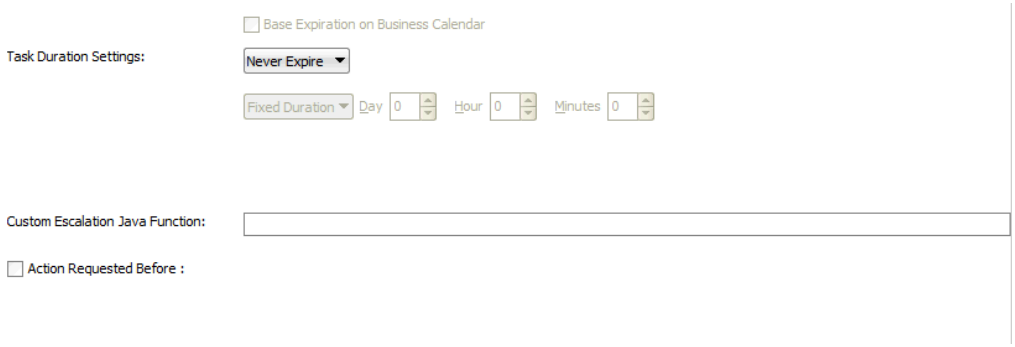
Escalating, Renewing, or Ending the Task

Figure 29-49 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** check box), 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 Specifying a Time Limit for Acting on a Task.

Figure 29-49 Human Task Editor — Deadlines Section



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 29-50:

Figure 29-50 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. 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.

How to Specify a Policy to Never Expire

You can specify for a task to never expire.

In the drop-down list in the **Deadlines** section, as shown in Figure 29-49, select **Never Expire** to specify a policy to never expire.

How to Specify 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 drop-down list of the **Deadlines** section, select **Expire after**, as shown in Figure 29-51.
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 29-51 indicates that the task expires in three days.

Figure 29-51 Expire After Policy

Note:

The escalation time is limited to future times that are before the year 2286. Using a value that is greater results in runtime errors. The technical limit of the future value is 9,999,999,999,999 milliseconds since January 1, 1970, 00:00:00 GMT.

How to Extend 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).

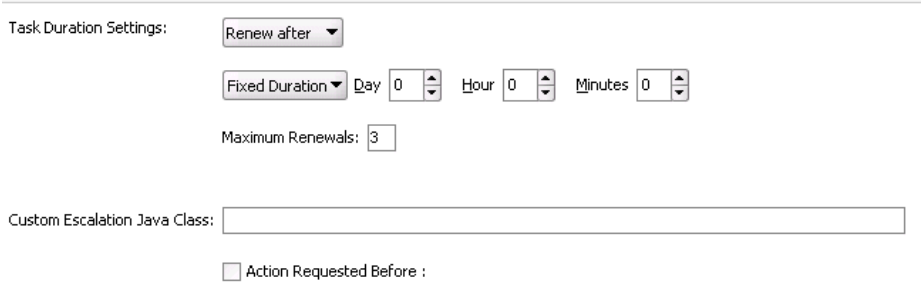
To extend an expiration policy period:

1. In the drop-down list of the **Deadlines** section, select **Renew after**, as shown in Figure 29-52.

2. Specify the maximum number of times to continue renewing this task.

   In Figure 29-52, 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 29-52 Renew After Policy



How to Escalate 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 drop-down list of the **Deadlines** section, select **Escalate after**, as shown in Figure 29-53.
2. Specify the following additional values. When both are set, the escalation policy is more restrictive.

   - **Maximum Escalation Levels**

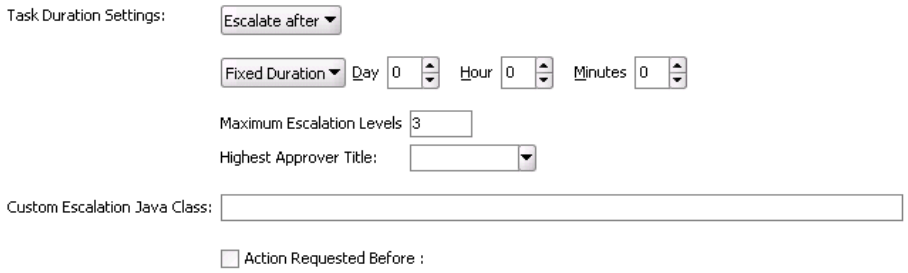
     Number of management levels to which to escalate the task. This field is required.

   - **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 29-53, 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 29-53 Escalate After Policy



How to Specify 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 Config** > **Task** tab.

   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 Custom Escalation Function.

How to Specify a Due Date

A due date indicates the date by which the task should be completed. 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 29-49. 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, filter 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** check box.
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 box 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 How To Create a ToDo Task.

Specifying Participant Notification Preferences

Figure 29-54 shows the **General** tab of the **Notification** section of the Human Task Editor (when fully expanded).

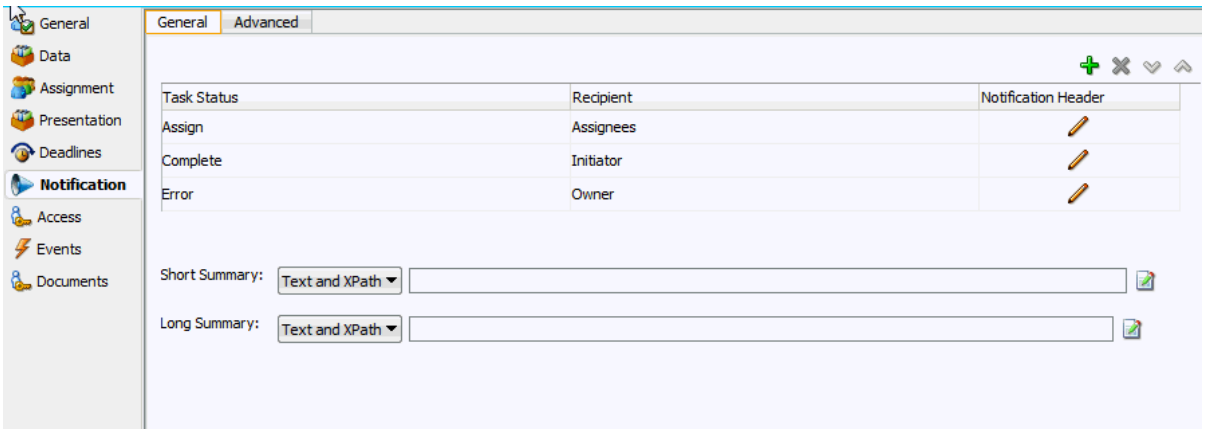
Notifications indicate when a user or group is assigned a task or informed that the status of the task has changed. Notifications can be sent through email, voice message, instant message, or SMS. Notifications are sent to different types of participants for different actions. Notifications are configured 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 29-54 Human Task Editor — General Tab of Notification Section



To specify participant notification preferences:

1. Click the **Notification** tab (displays as shown in Figure 29-54).

   Instructions for configuring the following subsections of the **General** tab of the **Notification** section are listed in Table 29-15.

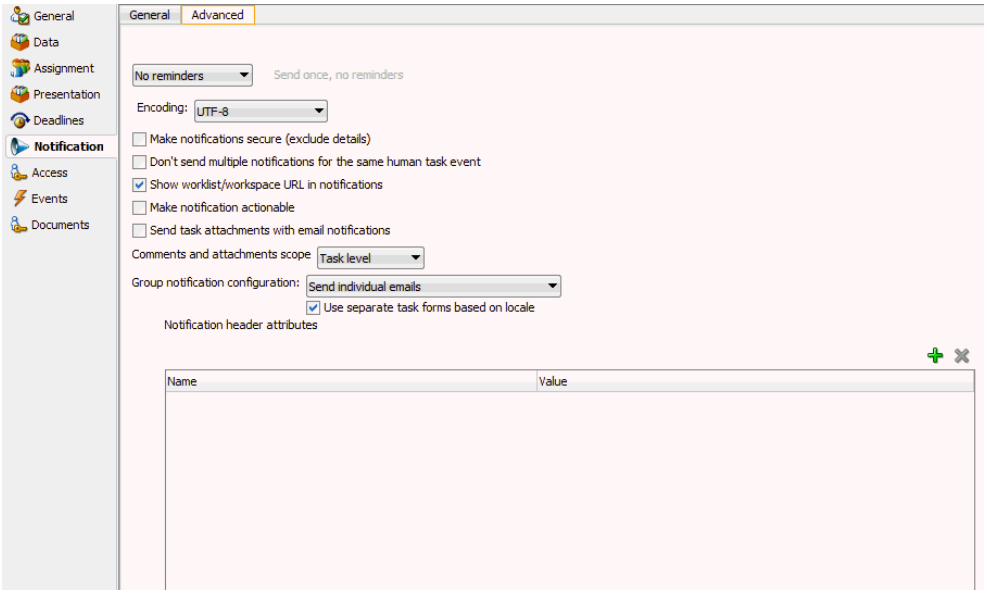
Table 29-15 Human Task Editor — General Tab of Notification Section

For This Subsection...	See...
Task Status **Recipient**	How to Notify Recipients of Changes to Task Status
Notification Header	How to Edit the Notification Message

   For information about the notification service, see Notifications from Human Workflow.

2. In the **Notification** section, click the **Advanced** tab. Figure 29-55 provides details.

Figure 29-55 Notification Section - Advanced Tab



Instructions for configuring the following subsections of the **Advanced** tab of the **Notification** section are listed in Table 29-16.

Table 29-16 Human Task Editor — Advanced Tab of Notification Section

For This Subsection...	See...
Reminders	How to Set Up Reminders
Encoding	How to Change the Character Set Encoding
Make notifications secure (exclude details)	How to Secure Notifications to Exclude Details
Show worklist URL in notifications	How to Display the URL in Notifications
Make notifications actionable	How to Make Email Messages Actionable
Send task attachments with email notifications	How to Send Task Attachments with Email Notifications
Group notification configuration	How to Send Email Notifications to Groups and Application Roles
Notification header attributes	How to Customize Notification Headers

How to Notify 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 box under the **Task will go from starting to final participant** icon in the **Assignment** section. For more information, see How to Configure the Error Assignee and Reviewers.

- **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 How to Configure the Notification Channel Preferences.

How to Edit the Notification Message

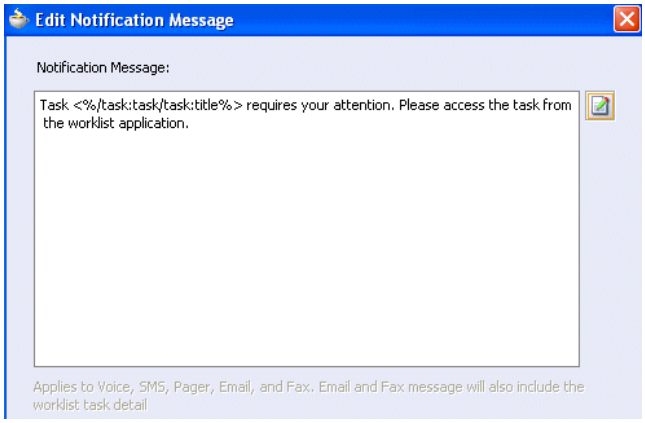
A default notification message is available for delivery to the selected recipient. If you want, you can modify the default message text.

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 box shown in Figure 29-56 appears.

Figure 29-56 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 Notifications from Human Workflow.

How to Set 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 How to Send Reminders.

How to Change 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.

How to Secure 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 How to Send Secure Notifications.

How to Display the Oracle BPM Worklist URL in Notifications

You can configure whether to display the Oracle BPM Worklist URL in email notification messages.

To display the Oracle BPM Worklist URL in notifications:

1. In the **Notification** section, click the **Advanced** tab.
2. Select the **Show worklist URL in notifications** check box to display the Oracle BPM Worklist URL in email notification messages. If this check box is not selected, the URL is not displayed.

How to Make 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 How to Send Actionable Messages.

   For more information about configuring outbound and inbound emails, see *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

How to Send 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**.

How to Send 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** check box 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.

     Note:

     – 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.

How to Customize 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.

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:

   - How to Send Inbound and Outbound Attachments
   - How to Create Custom Notification Headers
   - *Developing Applications with Oracle User Messaging Service*

Specifying Access Policies and Task Actions on Task Content

You can specify access rules on task content and actions to perform on that 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.

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`. as shown in the example below:

```
public Map<String, IPrivilege> getTaskVisibilityRules(IWorkflowContext context,
                                    String taskId)
   throws TaskMetadataServiceException;
```

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

Specifying User Privileges for Acting on Task Content

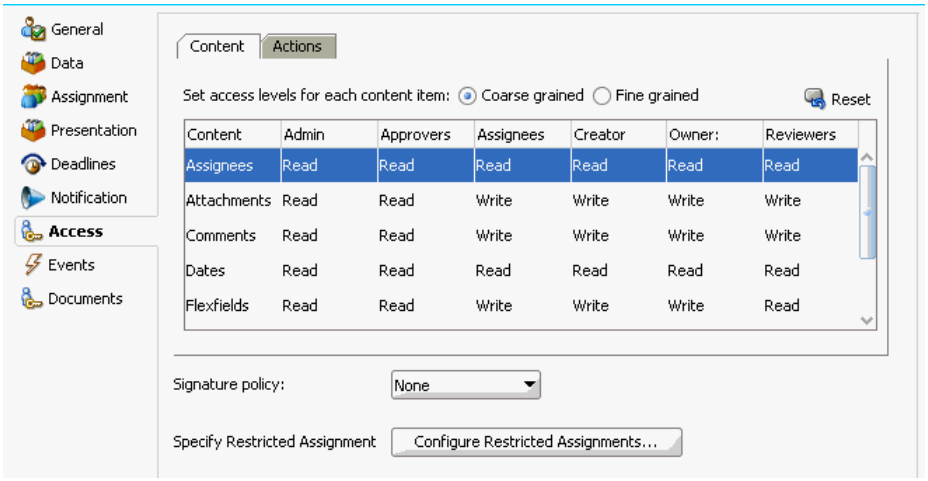
You can specify the privileges that specific users (such as the task creator or owner) have for acting on specific task content (such as a payload).

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 29-57.

Figure 29-57 Configure Task Content Access



4. Assign privileges (read, write, or no access) to users to act upon task content. 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 29-17 shows the maximum privilege each user has on task content.

Table 29-17 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
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 box appears as shown in Figure 29-57.

5. Select the method for displaying task content in this dialog box. 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.

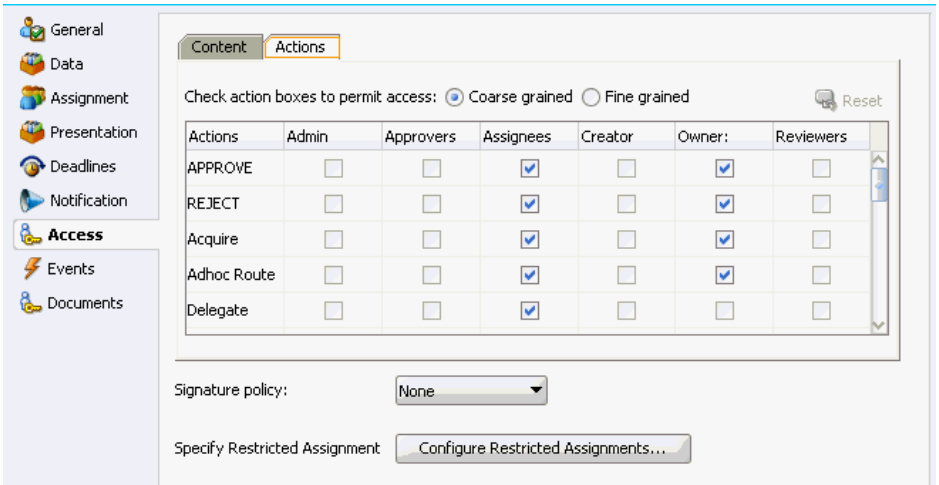
Specifying Actions for Acting Upon Tasks

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 box.

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 29-58.

Figure 29-58 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 box. 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).

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.

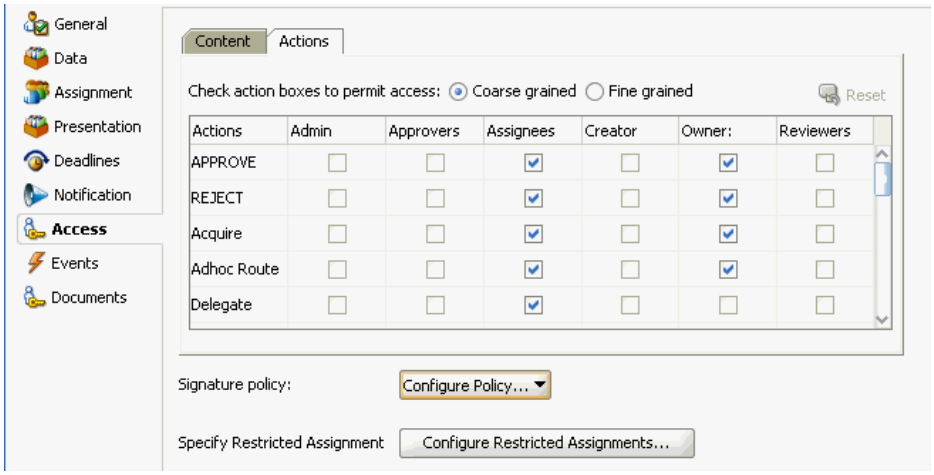
Note:

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 29-59.

Figure 29-59 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 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 Evidence Store Service and Digital Signatures.

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.

Specifying 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.

Note:

Certain functions, such as restricted task reassignment, are available only when a single task is selected. If multiple tasks that use restricted reassignment are selected, then the restricted reassignment algorithm is not invoked. In that case, the complete list of users gets returned as though restricted reassignment had not been specified.

How to Specify Restrictions on Task Assignments

To specify restrictions on task assignments:

1. In the **Access** section, click **Configure Restricted Assignments**.

   The Configure Restricted Assignment dialog box 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**.

Specifying Java or Business Event Callbacks

You can specify Java or business event callbacks.

Note:

If you implemented a callback, then the user callback implementation overrides any other form of restricted assignment. When you perform a search, the result only shows the users that the user callback returns.

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 state change 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).

If your Oracle JDeveloper installation is updated to include both the BPEL and BPM extensions, then the following content callbacks are also available for selection:

- **Comments Callback**

  Select if the callback class must be called to store the comments in a schema other than the `WFCOMMENTS` column. The callback class must implement the `oracle.bpel.services.workflow.callback.NotesStore` interface.

- **Attachment Call Back**

  Select if the callback class must be called to store the attachments in a schema other than the `WFATTACHMENT` table in the soa-infra schema. The callback class must implement the `oracle.bpel.services.workflow.callback.AttachmentStore` interface.

- **Validation Callback**

  Select if the callback class must be called to validate either the task or payload before updating, approving, and so on. The callback class must implement the `oracle.bpel.services.workflow.task.ITaskValidationCallback` interface.

2. See the following section based on the type of callback to perform.

   - Specifying Java Callbacks

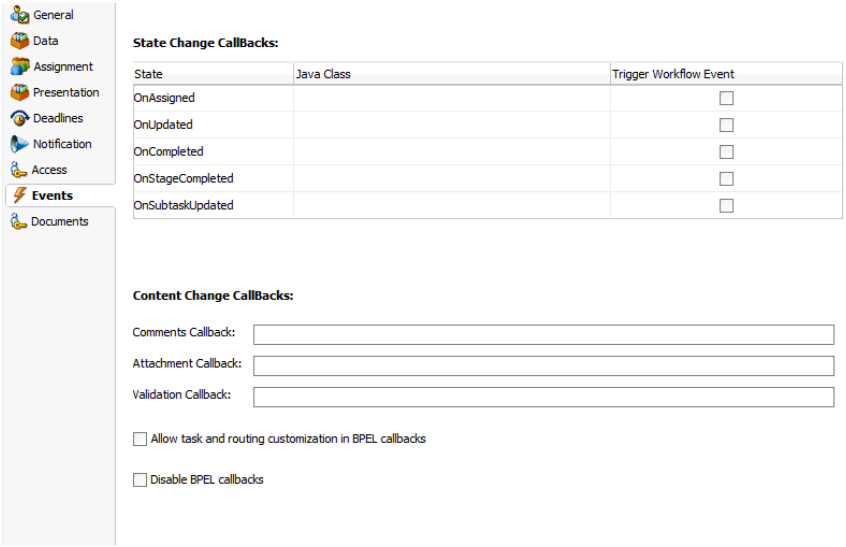
- Specifying Business Event Callbacks

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 29-60 provides details.

Figure 29-60 CallBack Details Dialog with Java Selected



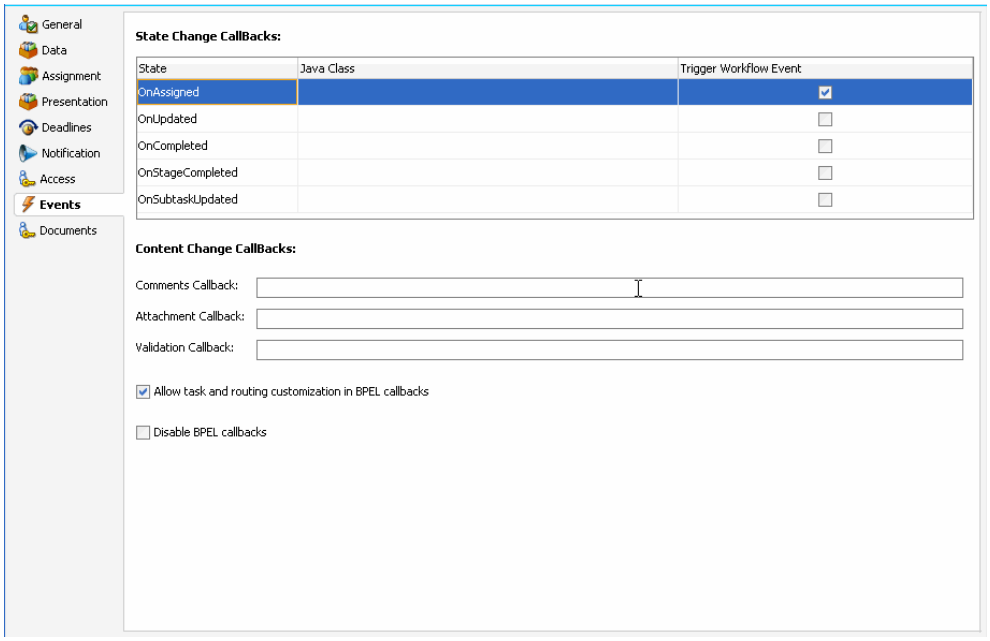
3. Click **OK**.

Specifying Business Event Callbacks

To specify business event callbacks:

1. In the **State** column of the **Events** section, select a task state. Leave the **Java Class** field empty.
2. Select the **Trigger Workflow Event** check box. This action disables the **Java Class** column, as shown in Figure 29-61. 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 29-61 CallBack Details Dialog with Business Events Selected



A pre-seeded, static event definition language (EDL) file (*JDev_Home* `\jdeveloper\integration\seed\soa\shared\workflow\HumanTaskEvent.edl`) provides the list of available business events to which to subscribe. These business events correspond to the callbacks you select in the Callback Details dialog box. 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`.

---

3. Create an Oracle Mediator service component in the same or a different SOA composite application that can subscribe to the event.
4. In the **Template** list during Oracle Mediator creation, select **Subscribe to Events**.
5. Click the **Add** icon to subscribe to a new event.
6. To the right of the **Event Definition** field, click the **Browse** icon to select the EDL file.

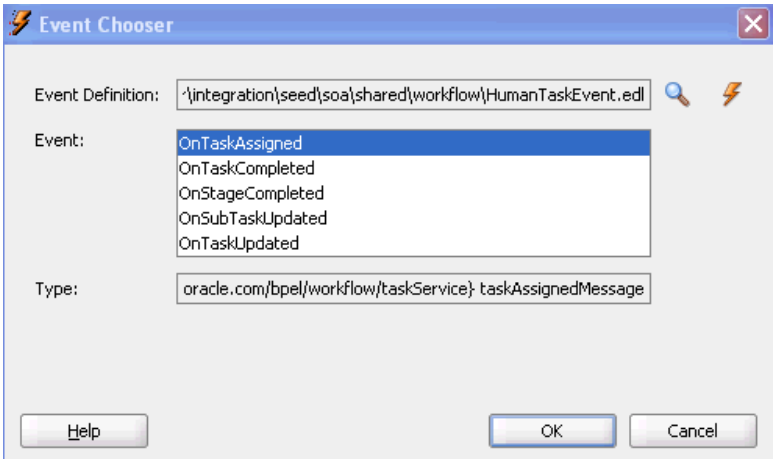
   The SOA Resource Browser dialog box appears.
7. Select the previously created file-based MDS connection.
8. From the list at the top, select **Resource Palette**.
9. Select **SOA** > **Shared** > **Workflow** > **HumanTaskEvent.edl**.

   Click **OK**.

The Event Chooser is now populated with EDL file business events available for selection.

10. In the **Event** field, select the event to which to subscribe. Figure 29-62 provides details.

Figure 29-62 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.

11. If the EDL file was *not* selected from the file-based MDS connection, accept to import the dependent XSD files when prompted, and click **OK**. If the EDL file was selected from the file-based MDS connection, you are not prompted.

    The Oracle Mediator service component is now populated with the business event to which to subscribe. You can also subscribe to other business events defined in the same EDL file now or at a later time.

See the following documentation for additional details about business events and callbacks:

- Using Business Events and the Event Delivery Network for specific details about business events
- Sample workflow-116-WorkflowEventCallback, which is available with the Oracle SOA Suite samples.

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** check box.
2. Return to Oracle BPEL Designer.
3. Open the task activity dialog box.
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 Invoking BPEL Callbacks.

How to Disable 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** check box 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** check box.
2. Click **OK**.

30

Designing Task Forms for Human Tasks

This chapter describes how developers can design and customize task forms for human tasks by using ADF task flows in Oracle JDeveloper. Human tasks enable 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.

This chapter includes the following sections:

For information about troubleshooting human workflow issues, see section "Human Workflow Troubleshooting" of *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

Introduction to the Task Form

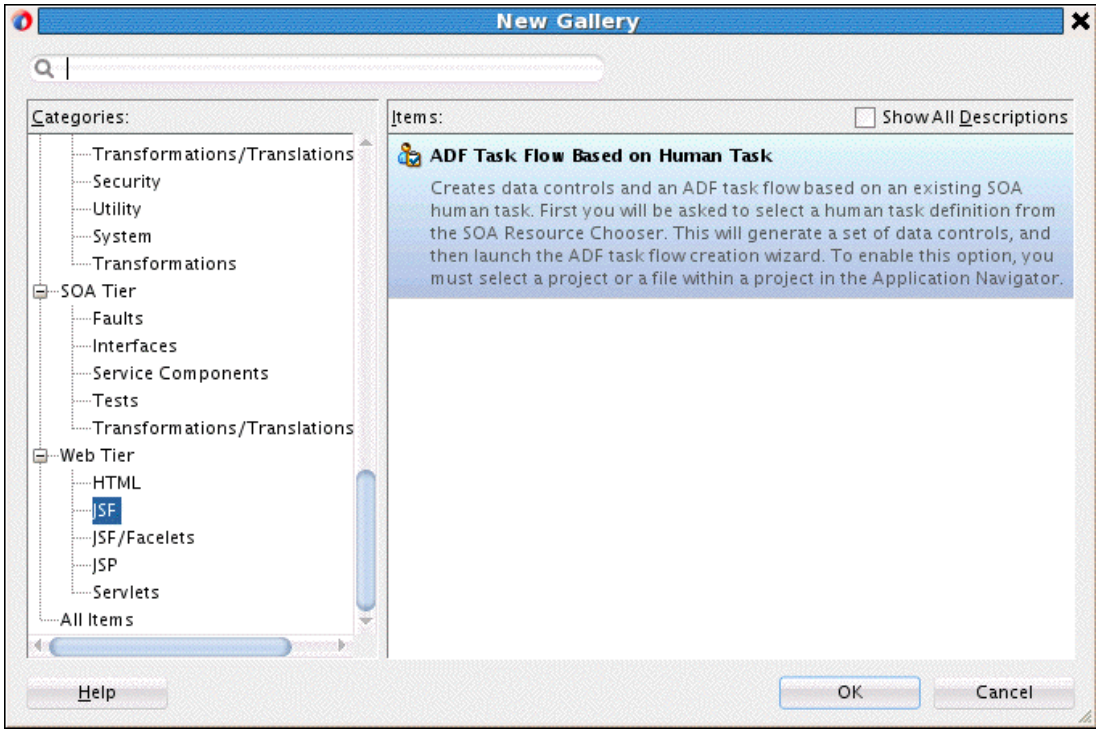
If your SOA composite includes a human task, then you need a way for users to interact with the task. The integrated development environment of Oracle SOA Suite includes Oracle Application Development Framework (Oracle ADF) for this purpose. With Oracle ADF, you can design a task form that depicts the 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. 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 30-1 shows the Oracle JDeveloper **ADF Task Flow Based on Human Task** option where you start creating a task form.

Figure 30-1 ADF Task Flow Based on a Human Task, in Oracle JDeveloper



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. The following example shows a sample header:

```
<af:outputText value="#{bindings.createdDate.inputValue}"
                        id="ot15">
        <f:convertDateTime type="#{pageFlowScope.dt}"
                           timeZone="#{pageFlowScope.tz}"
                           dateStyle="#{pageFlowScope.df}"
                           timeStyle="#{pageFlowScope.tf}"/>
        </af:outputText>
```

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. For more information, see "Getting Started with ADF Task Flows" in *Developing Fusion Web Applications with Oracle Application Development Framework*.

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.

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 Creating 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 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 How To Create an ADF Task Flow Based on a Human Task, for more information.

An ADF task form does not validate user inputs. The only validation that is done is to check that mandatory inputs have values. You should review your task forms and add additional validators as needed.

How To Create an ADF Task Flow from the Human Task Editor

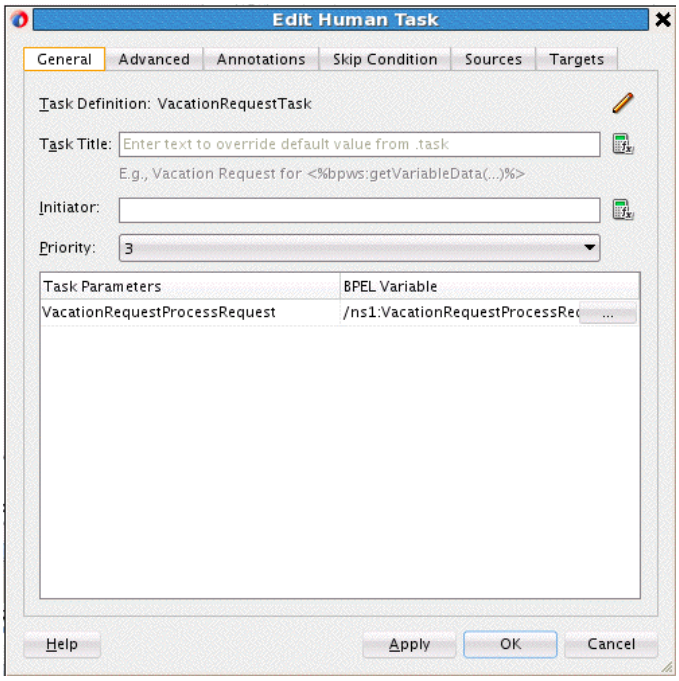
The `.task` file that specifies the human task is easily associated with the task flow when the two are located in the same application.

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**.

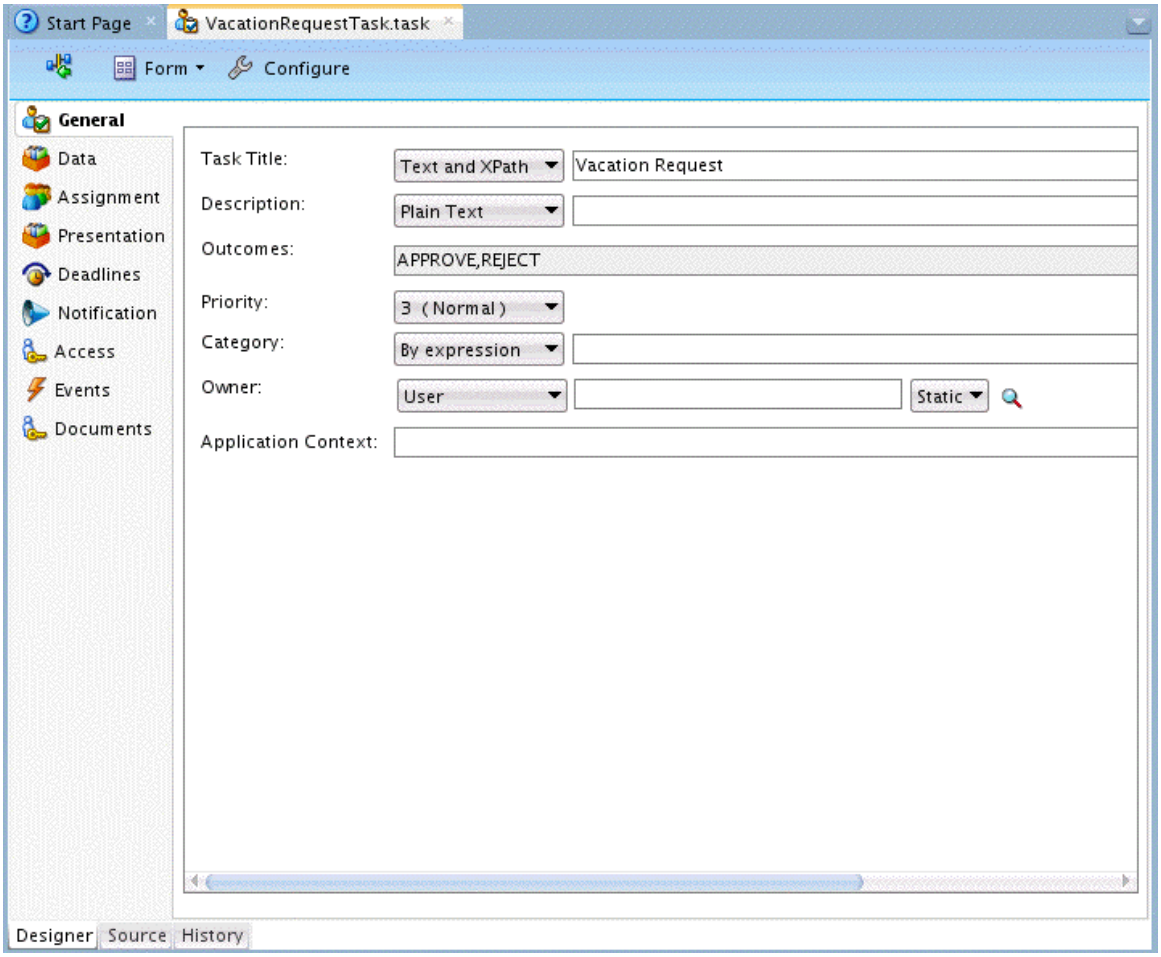
   Figure 30-2 shows the Human Task dialog.

Figure 30-2 Editing a Human Task



3. In the **.task** tab (shown in Figure 30-3), click **Form** and select **Auto-Generate Task Form**.

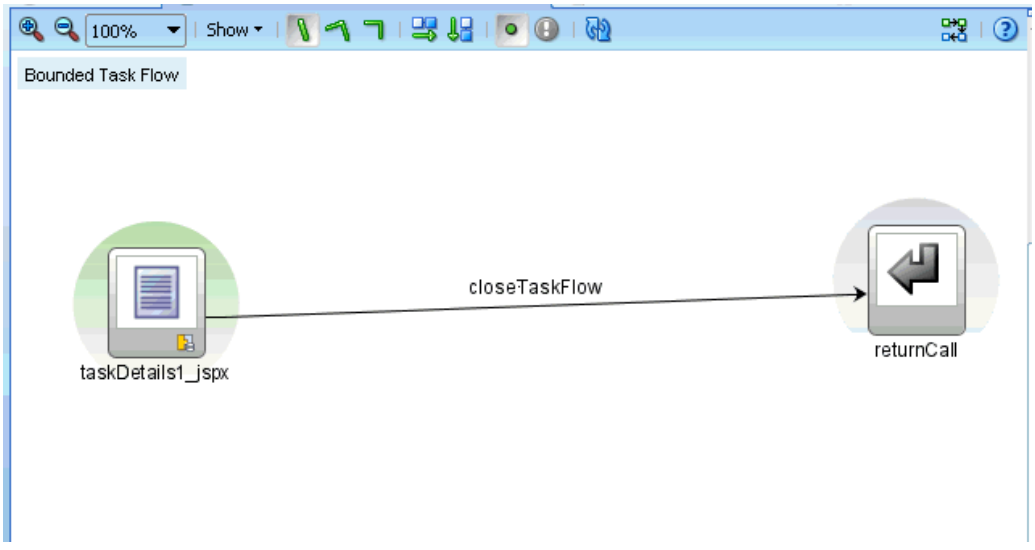
Figure 30-3 Creating a Task Flow from the Human Task Editor



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 30-4.

Figure 30-4 The taskDetails1_jspx Icon



The task flow and task form are complete and ready to be deployed.

How To Create an ADF Task Flow Based on a Human Task

The **ADF Task Flow Based on Human Task** option (shown in Figure 30-1) 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** > **Custom 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 System** 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 **SOA-MDS** 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 30-4. The task flow has a view, a control flow, and a task return.

To continue creating the task form, see the following:

- How To Create a Task Form Using the Complete Task with Payload Drop Handler.
- How To Create Task Form Regions Using Individual Drop Handlers.

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

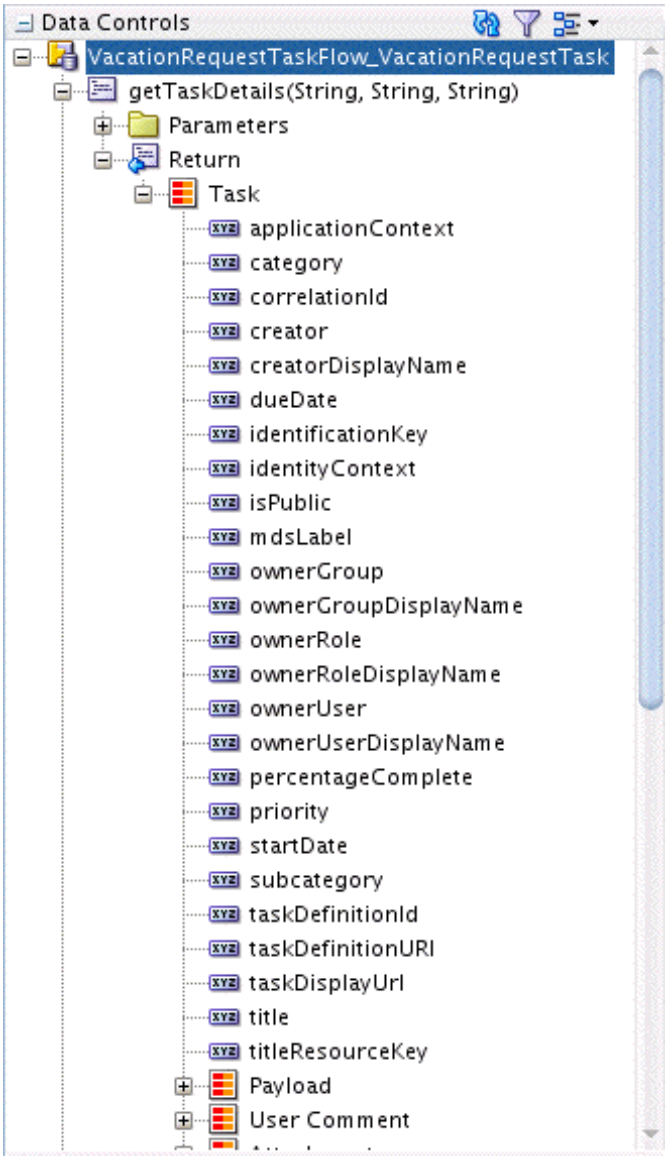
With an ADF task flow based on a human task, the task flow application has task data controls that wire the task form with the workflow services. The data controls provide the following:

- 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 Applications window, as shown in Figure 30-5.

Figure 30-5 The Task Collection in the Data Controls Panel



The data controls for the task (represented by the **Task** node in Figure 30-5) have drop handlers to render the task form. See Creating a Task Form, for more information.

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, 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).

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 How To Create an Autogenerated Task Form.
- For how to use the **Launch Task Form Wizard** option, see How To Create a Task Form Using the Custom Task Form Wizard.
- For how to use human task drop handlers, see the following:
  - How To Create a Task Form Using the Complete Task with Payload Drop Handler
  - How To Create Task Form Regions Using Individual Drop Handlers
  - How To Add the Payload to the Task Form

Note:

A task form name 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).

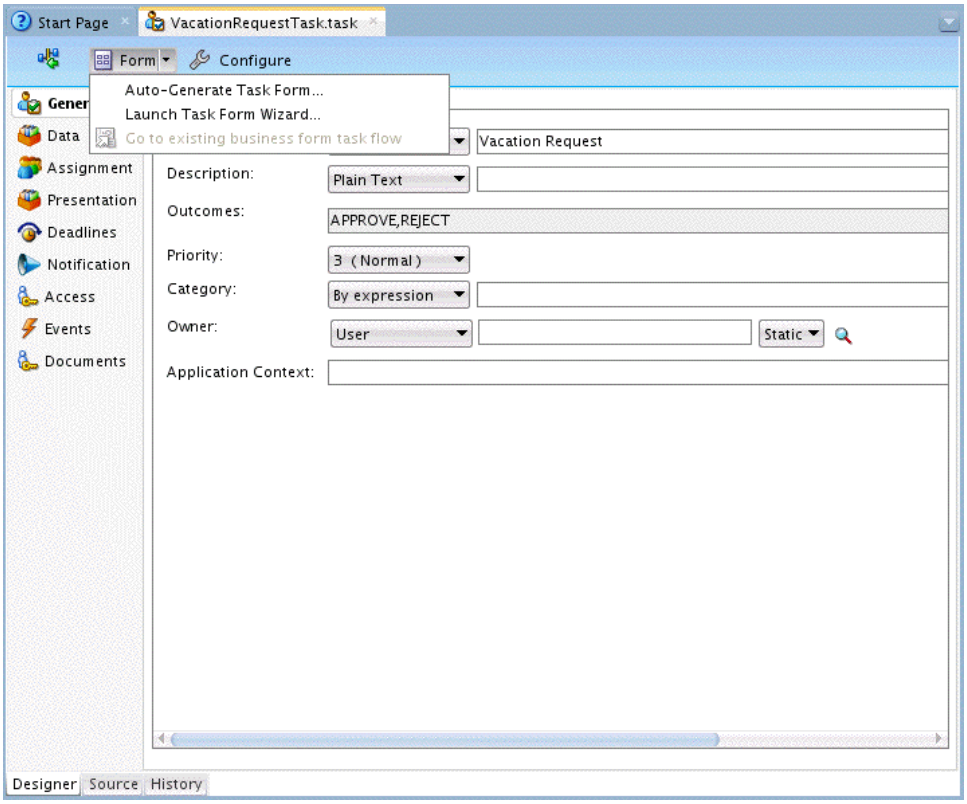
How To Create an Autogenerated Task Form

Autogenerating a task form opens a default template that you can then modify.

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 30-6.

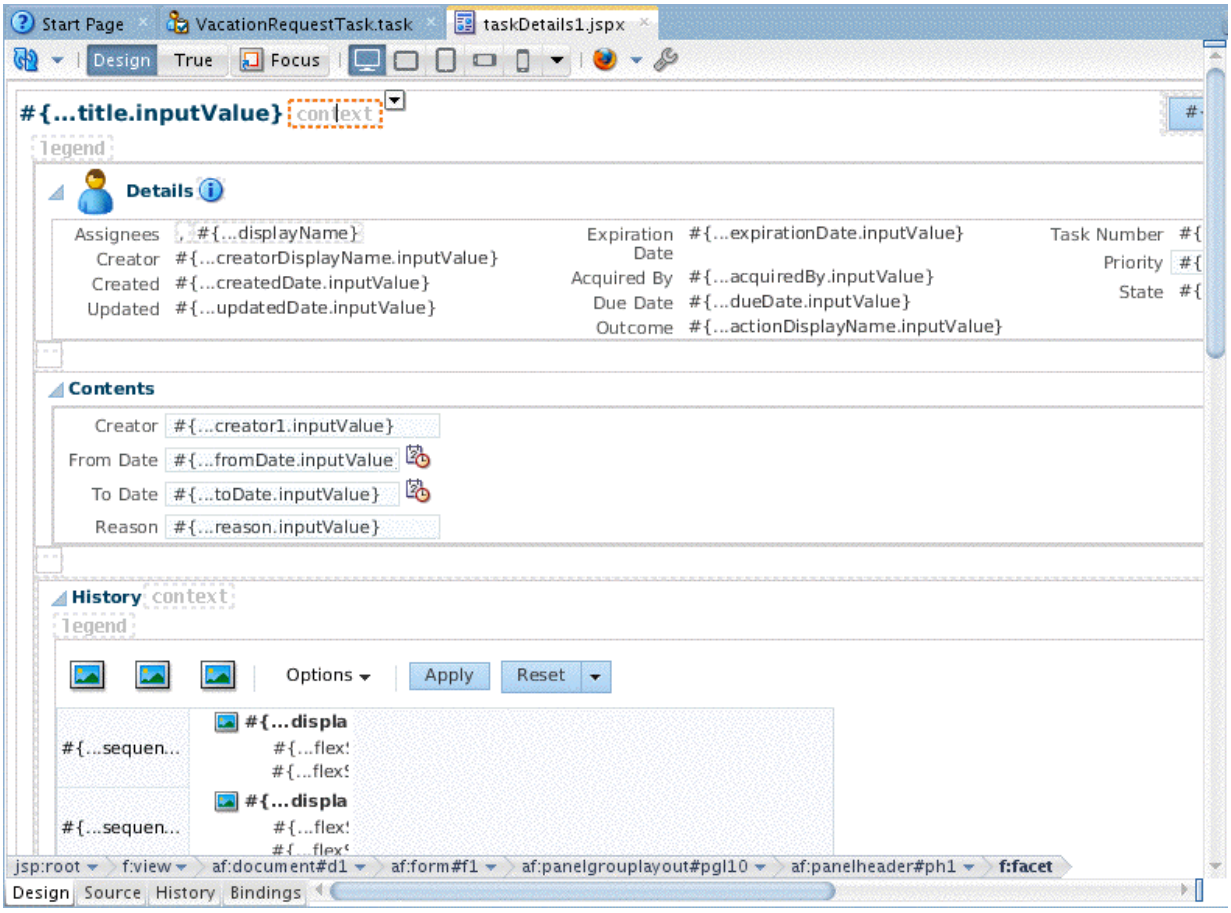
Figure 30-6 Creating a Task Form



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 30-7.

Figure 30-7 Autogenerated Task Form for ApprovalHumanTask



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 How To Create a Task Form Using the Custom Task Form Wizard, you select **Custom** in the Name and Definition page of the Custom Task Form Wizard and select the library and `.jspx` template.

As a prerequisite, you first must register the library JAR file in Oracle JDeveloper.

To create the library JAR file for custom page templates:

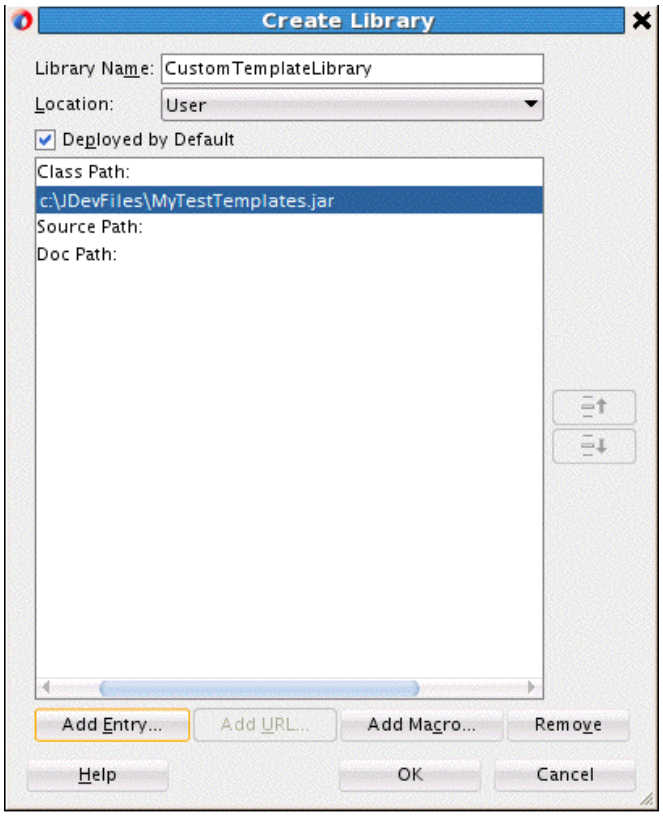
1. From the **Tools** menu, select **Manage Libraries**.
2. Click **New**.

   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 30-8 provides details.

Figure 30-8 Custom Library JAR File



5. Select the **Deployed by Default** check box.
6. Click **OK**.

   When you run the Custom Task Form wizard, you select the Custom radio button on the Name and Definition page, and select the library and template that you registered.

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`

  In the Name and Definition page of the Custom Task Flow wizard, select Packaged, then select either **Default** or **Tabbed**.

- Custom page templates that you define. In the Name and Definition page of the Custom Task Flow wizard, select **Custom**, then select the library name and the template name.

  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 to be displayed 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.

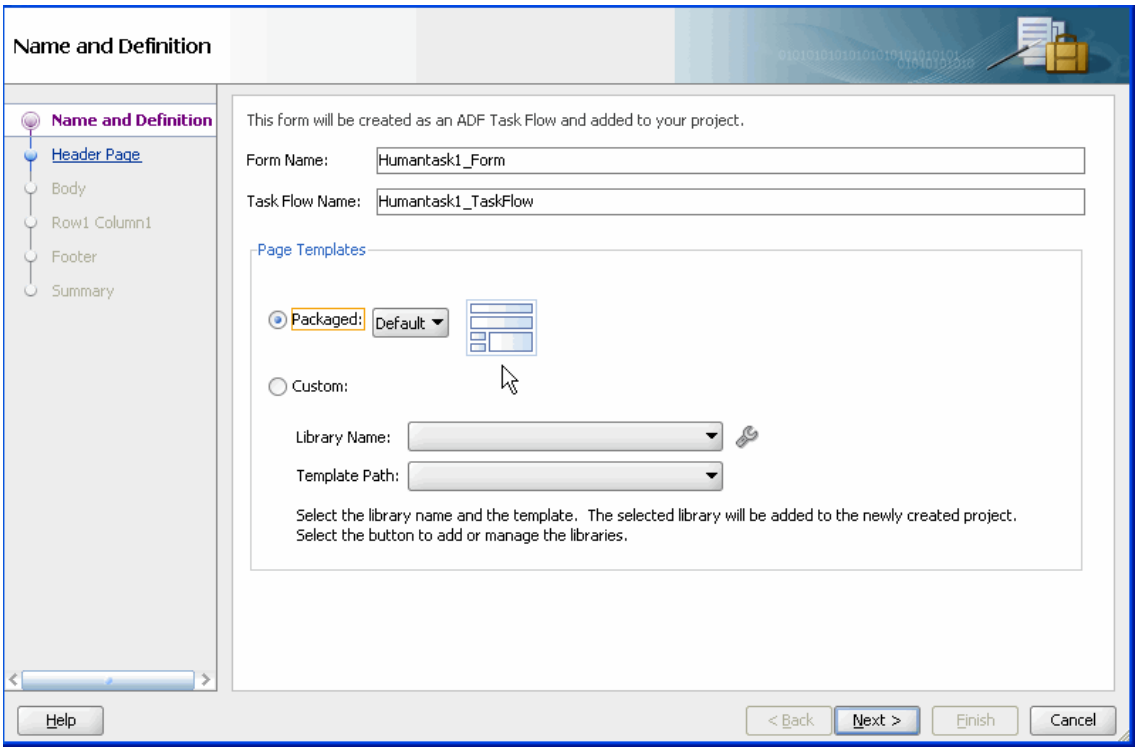
For more information about facets, see *Developing Web User Interfaces with Oracle ADF Faces*.

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 **Form** and select **Launch Task Form Wizard**.
4. Provide a project name and a directory path (or use the default), and click **OK**. The Custom Form Wizard displays the Name and Definition screen as shown in Figure 30-9.

Figure 30-9 Custom Task Form Wizard: Form Name and Definition



5. In the **Form Name** field, provide the name of the form (`.jspx` file) that is to be generated at the end of the wizard. If you do not provide a name, then the default name, `Humantask`*`number`*`_Form`, is provided. Ensure that valid characters are used in the name. Spaces are not permitted.

6. Specify the **Task Flow Name**, that is, 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.

7. In the **Page Templates** section, select either:

   - **Packaged**: Select this to use one of the default page templates, then select the particular template from the list.
   - **Custom**: Select the library and template. If no library is listed, click **Manage Libraries** and follow the instructions in How to Register the Library JAR File for Custom Page Templates.

   Click **Next**. The Header page appears.

8. On the Header page, shown in Figure 30-10, perform the following procedures and click **Next**.

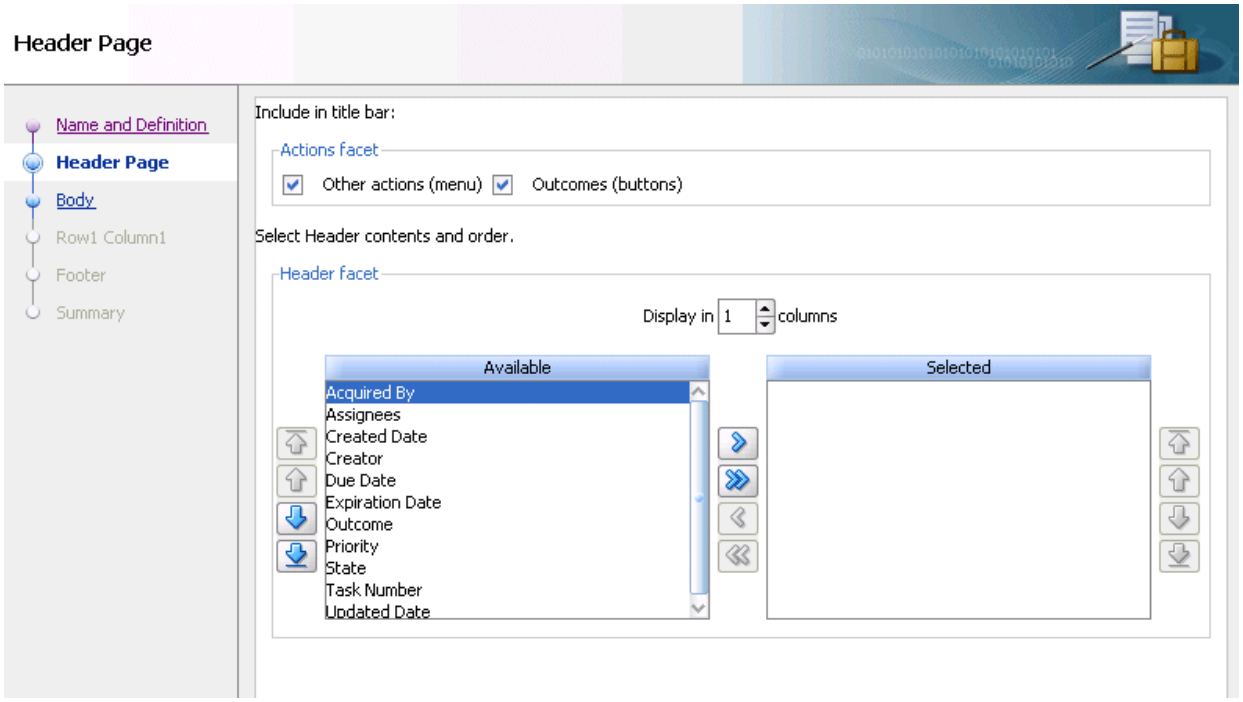
   - In the **Actions facet** section, select the options to include in the title bar of the task form:

     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.

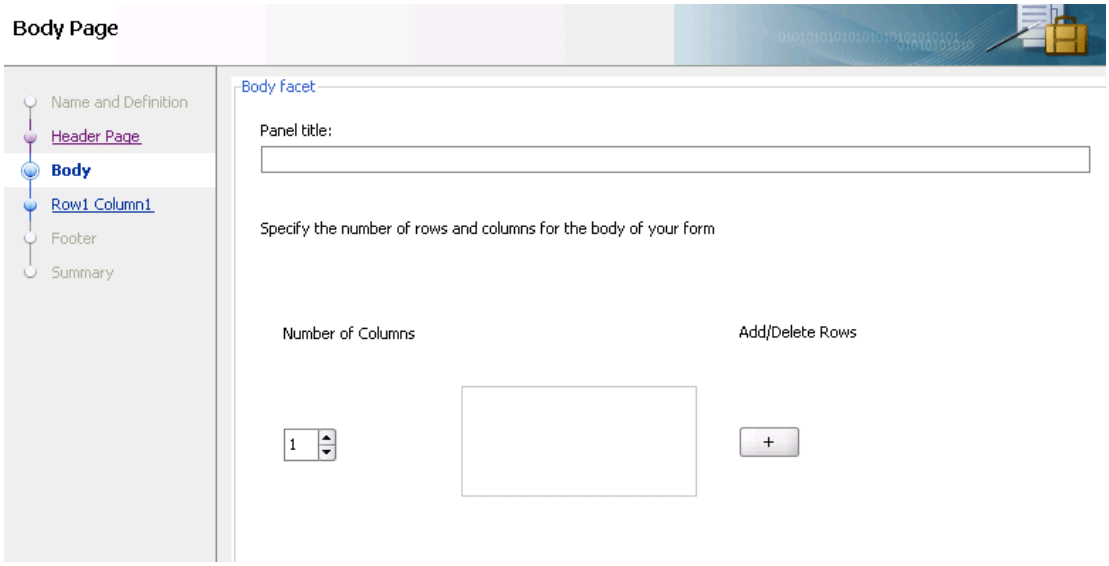
- In the **Header facet** section, enter the number of display columns. If you want each header label to display in its own column, then enter the same number as the number of headers you move into the **Selected** list. If you enter 1, but select 7 headers, all 7 headers appear in one column.
- Move header labels into the **Selected** list and reorder them as needed.

Figure 30-10 Custom Task Form Wizard: Setting Up the Header



9. On the Body page, shown in Figure 30-11, 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 30-11 Custom Task Form Wizard: Setting Up the Body

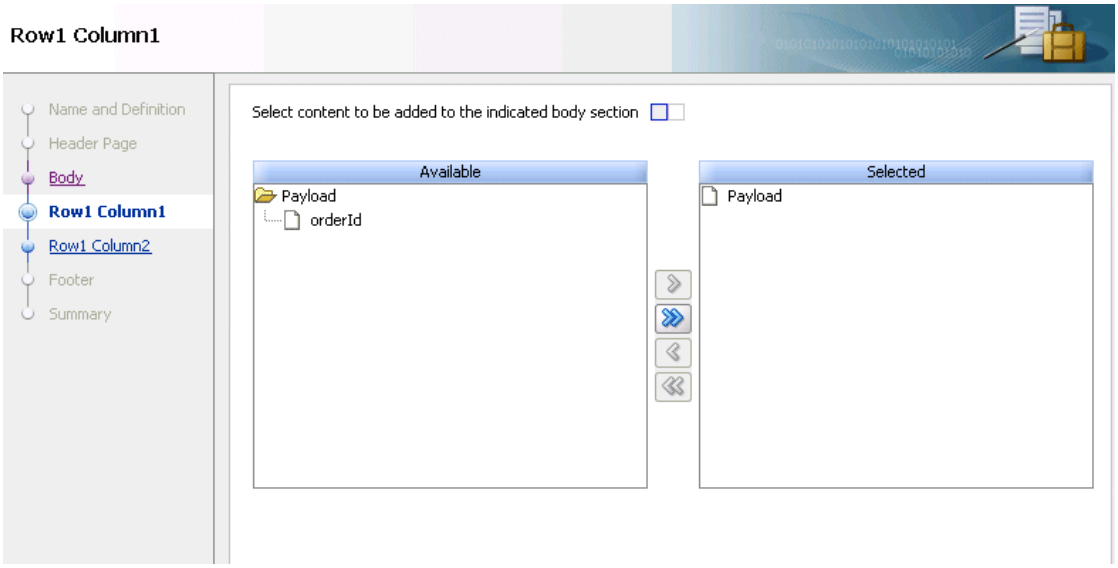


Note:

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 30-12, move all or part of the payload to the **Selected** list and click **Next**.

Figure 30-12 Custom Task Form Wizard: Selecting the Body Fields

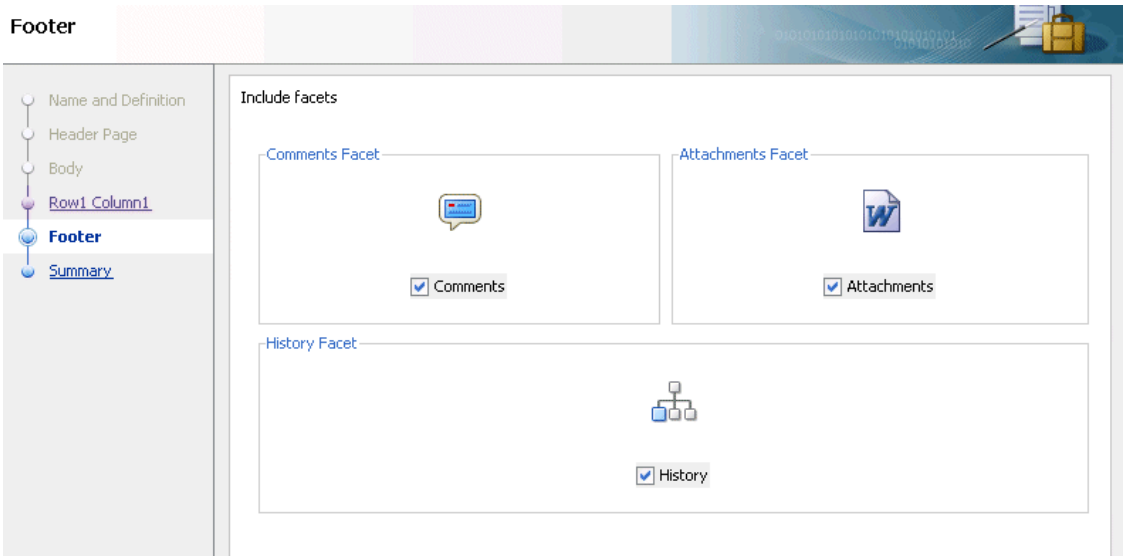


11. For any Row*n* Column*n* page after Row1 Column1, repeat Step 10 and click **Next**.

    The Footer page that displays is based upon the page template you selected on the Name and Definition page in Step 6 (either Default Page Template or Custom Page Template).

If you selected Default Page Template, the Footer page shown in Figure 30-13 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.

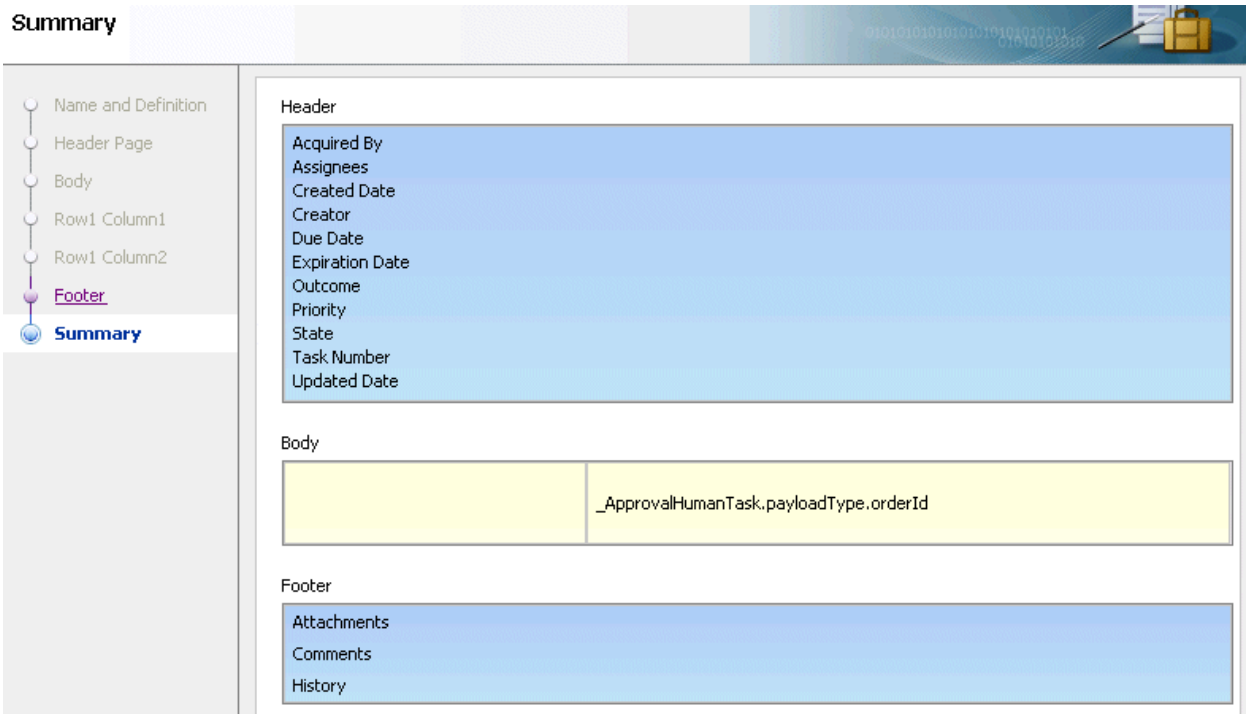
Figure 30-13 Custom Task Form Wizard: Selecting the Footer Fields for the Default Page Template



12. On the Summary page, shown in Figure 30-14, inspect your selections. Click **Back** to make changes or click **Finish**.

    This form is created as an ADF task flow and added to the project.

Figure 30-14 Custom Task Form Wizard: Summary



The Designer initializes and the *form_name*.jspx tab is displayed, as shown in Figure 30-15 (upper part of tab) and Figure 30-16 (lower part of tab).

Figure 30-15 Custom Task Form (Upper Part of Tab)

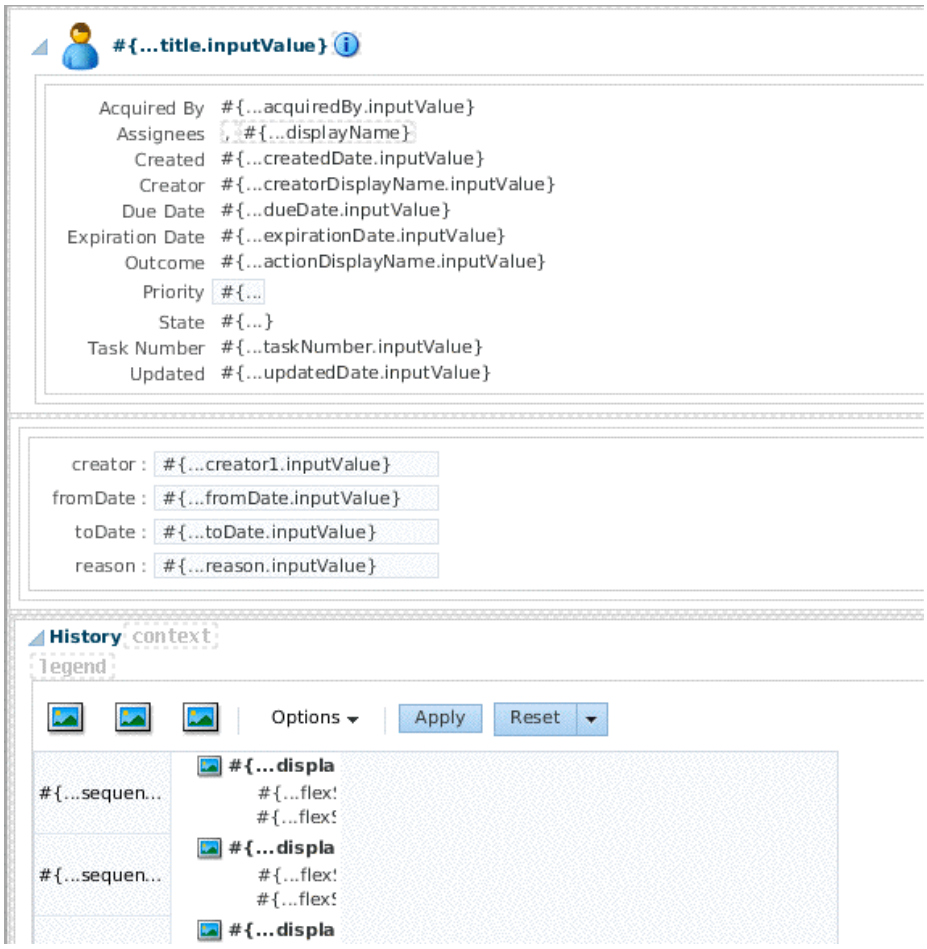
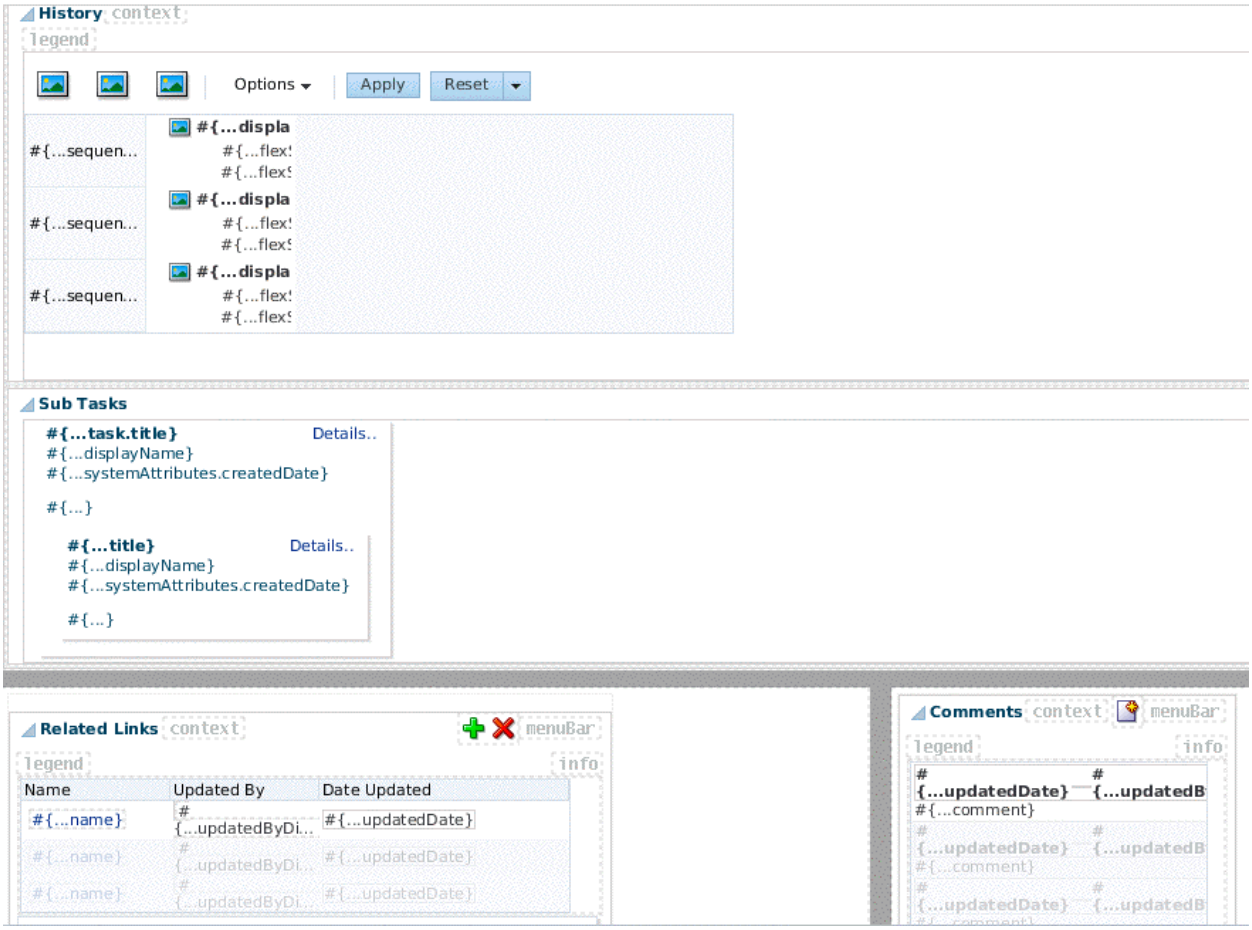


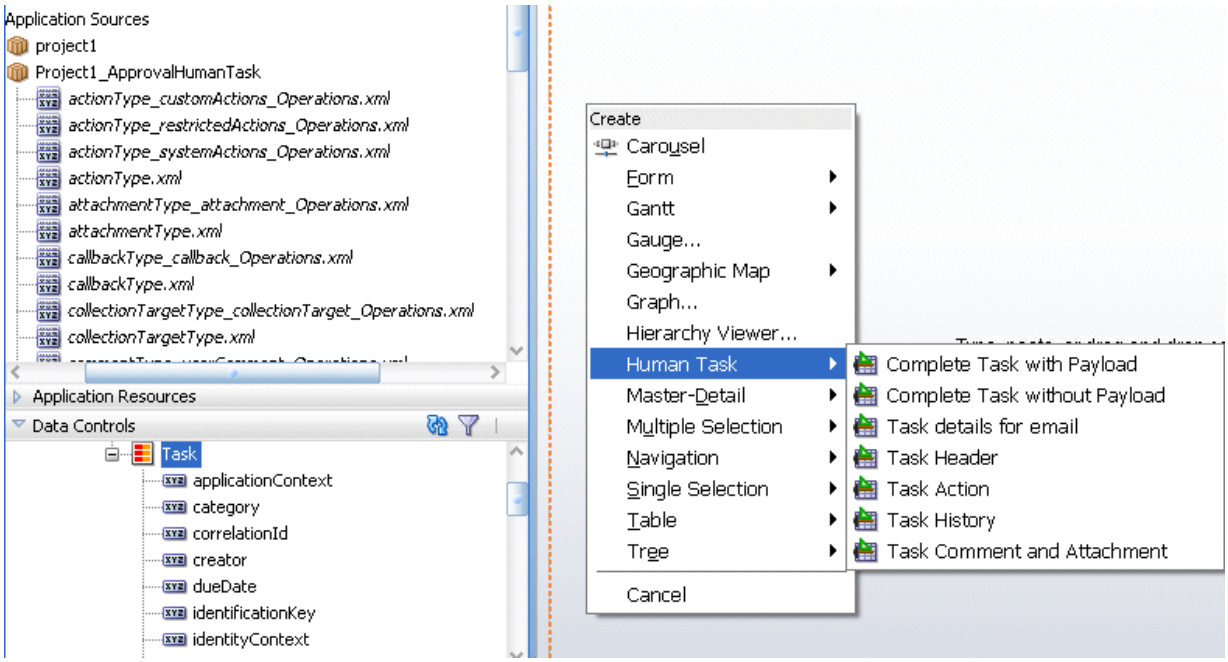
Figure 30-16 Custom Task Form (Lower Part of Tab)



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 30-17.

Figure 30-17 Human Task Drop Handlers for Creating the Task Form

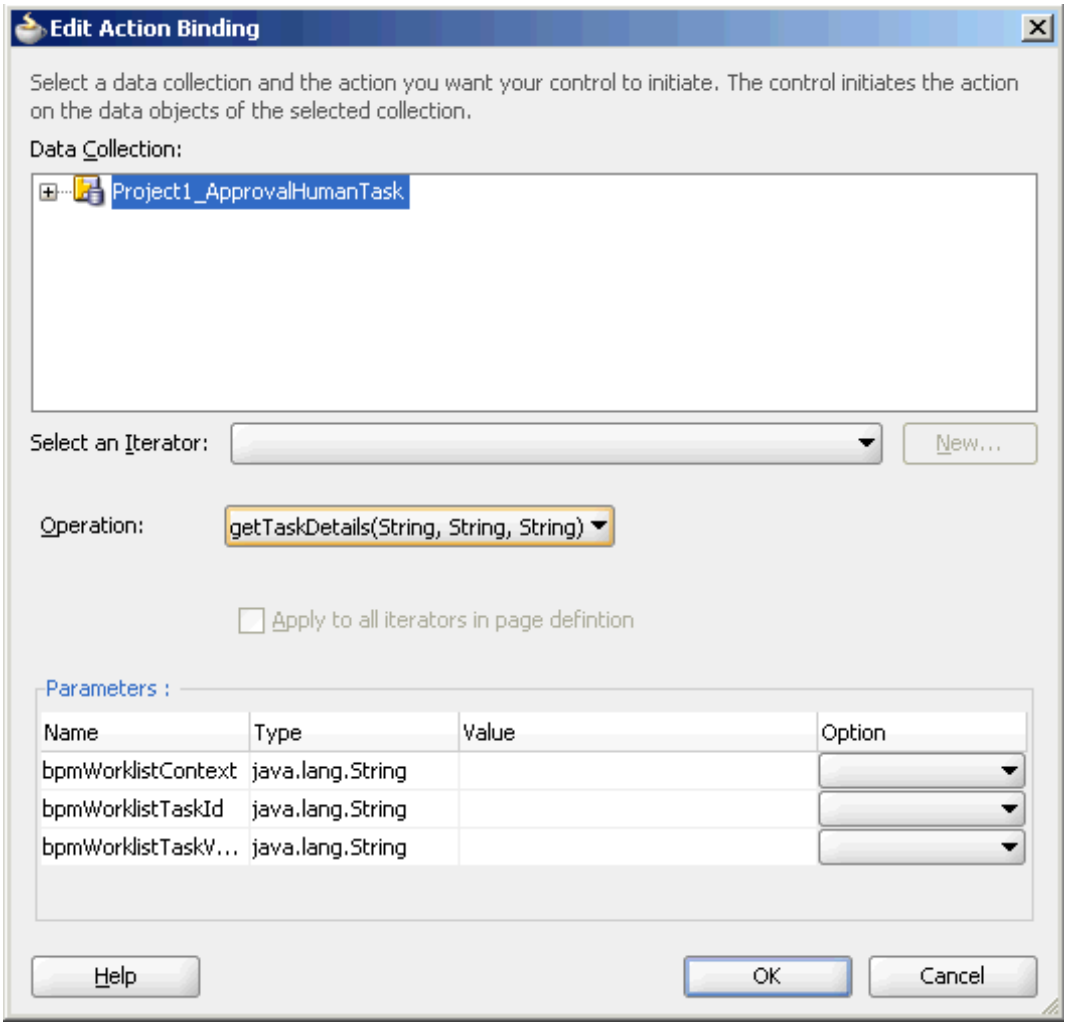


Other ADF drop handlers—for forms, tables, trees, and so on (shown in Figure 30-17)—can also be used to create task forms. See *Developing Fusion Web Applications with Oracle Application Development Framework* for more information about standard ADF drop handlers .

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

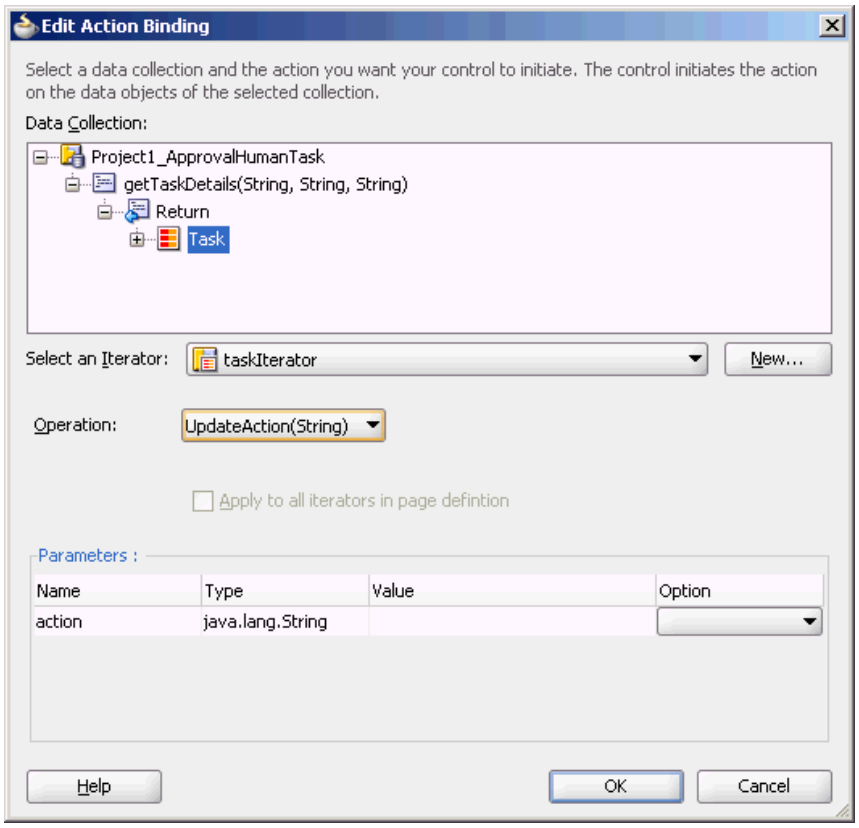
1. In the designer, double-click **taskDetails1_jspx**.
2. In the Create JSF Page dialog, provide a file name and directory information (or accept the defaults) and click **OK**.
3. In the **Data Controls** panel of the Applications window, expand the human task node, then the **getTaskDetails** node, 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 30-18, click **OK**.

Figure 30-18 Edit the Action Binding



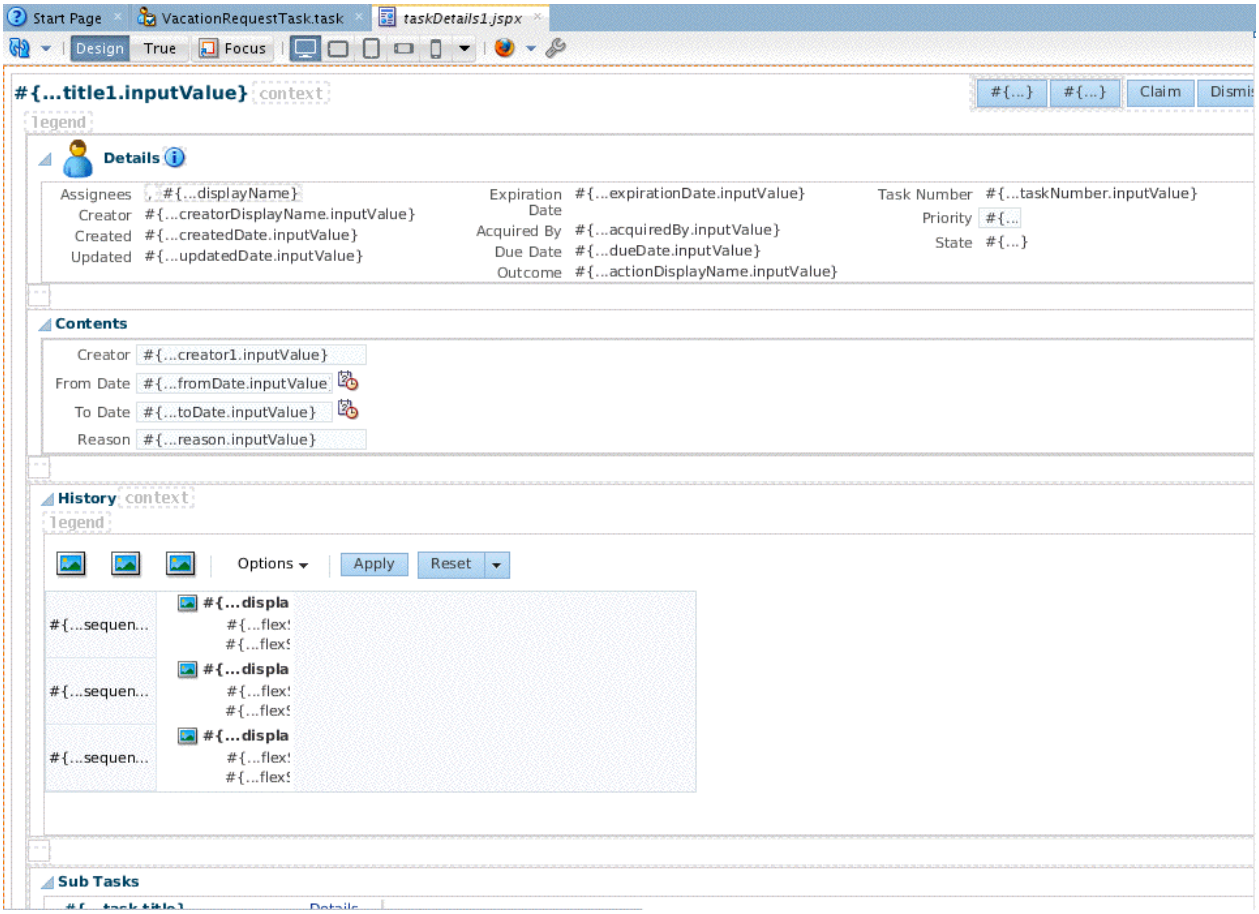
7. In the next Edit Action Binding dialog, the data collection is selected, as shown in Figure 30-19; click **OK**.

Figure 30-19 Select the Data Collection and Action



The task form is displayed, as shown in Figure 30-20.

Figure 30-20 Task Form



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 30-21.

Figure 30-21 Task Form for Email Notification

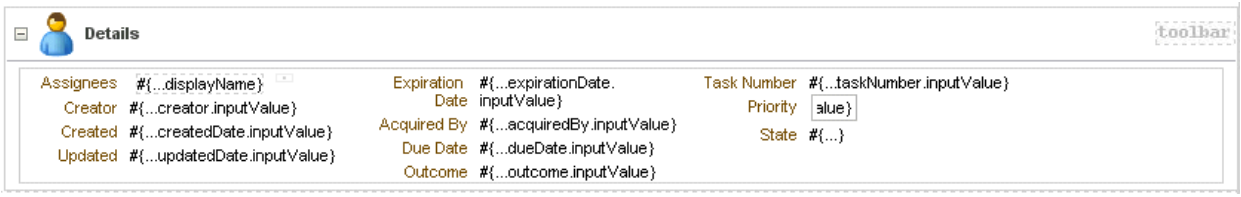
See Creating an Email Notification , for more information.

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.

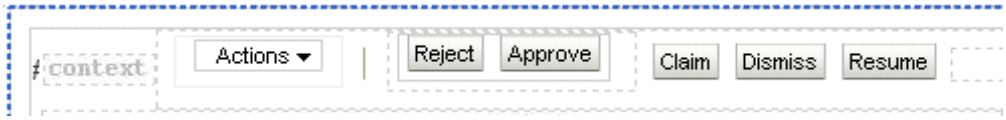
Figure 30-22 shows an example of header information.

Figure 30-22 Header Information



Buttons for task actions are also created in the header, as shown in Figure 30-23.

Figure 30-23 Task Header: Task Action Buttons



Task Actions

The following task actions appear from the **Actions** dropdown list or as buttons. The tasks that appear depend on the state of the task (assigned, completed, and so on) and the privileges that are granted to the user viewing the task. For example, when a task is assigned to a group, only the **Claim** button appears. After the task is claimed, other actions such as **Reject** and **Approve** appear.

The first three custom actions appear on the task form as buttons: **Claim**, **Dismiss**, and **Resume**. Only those buttons applicable to the task appear. Other actions are displayed

under the **Actions** list, starting with **Request for Information**, **Reassign**, and **Route**. Systems actions—**Withdraw**, **Pushback**, **Escalate**, **Release**, **Suspend**, and **Renew**—follow the custom actions, followed by the **Save** button. These actions require no further dialog to complete.

- **Claim**—A task that is assigned to a group or multiple users must be claimed first. **Claim** is the only action available in the **Task Action** list for group or multiuser assignments. After a task is claimed, all applicable actions are listed.

  Note:

  – If an FYI task is sent to multiple users, a user must first select the **Claim** button to claim the task before they can dismiss it.

  – Pushback is designed to work with single approvers and not with group votes. Pushback from a stage with group vote (or parallel) scenario to another stage is not allowed. Similarly, you cannot push back from a single assignee to a group vote (or parallel) scenario.

- **Dismiss**—This action is used for a task that requires the person acting on the task to acknowledge receipt, but not take any action (such as 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. **Note:** Pushback is designed to work with single approvers and not with group votes. Pushback from a stage with group vote (or parallel) scenario to another stage is not allowed. Similarly, you cannot push back from a single assignee to a group vote (or parallel) scenario.

- **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 . 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 32-19 and Figure 32-20 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.

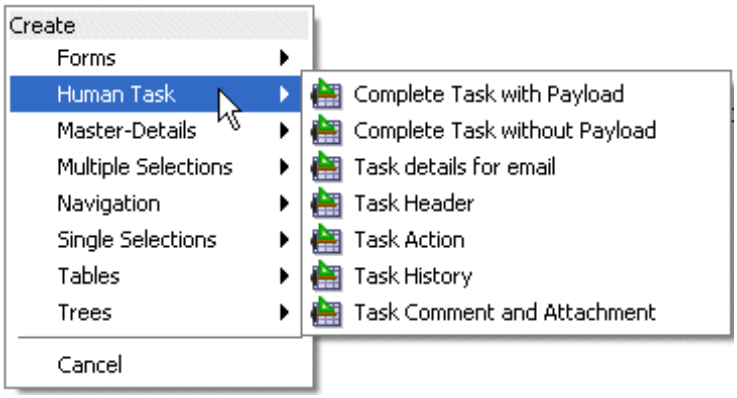
Figure 30-24 shows an example of the comments and attachments region.

Figure 30-24 Comments and Attachment Region

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 30-25.

Figure 30-25 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 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 Introduction to the Human Workflow Tutorial for more information.

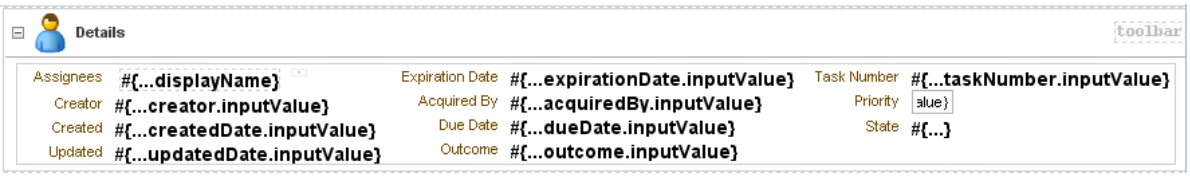
To create task form regions using individual drop handlers:

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 Applications window, 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 30-26, and header details, as shown in Figure 30-27.

Figure 30-26 Designing the Task Form: Buttons for Task Actions

Figure 30-27 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 How To Add the Payload to the Task Form.

How To Add the Payload to the Task Form

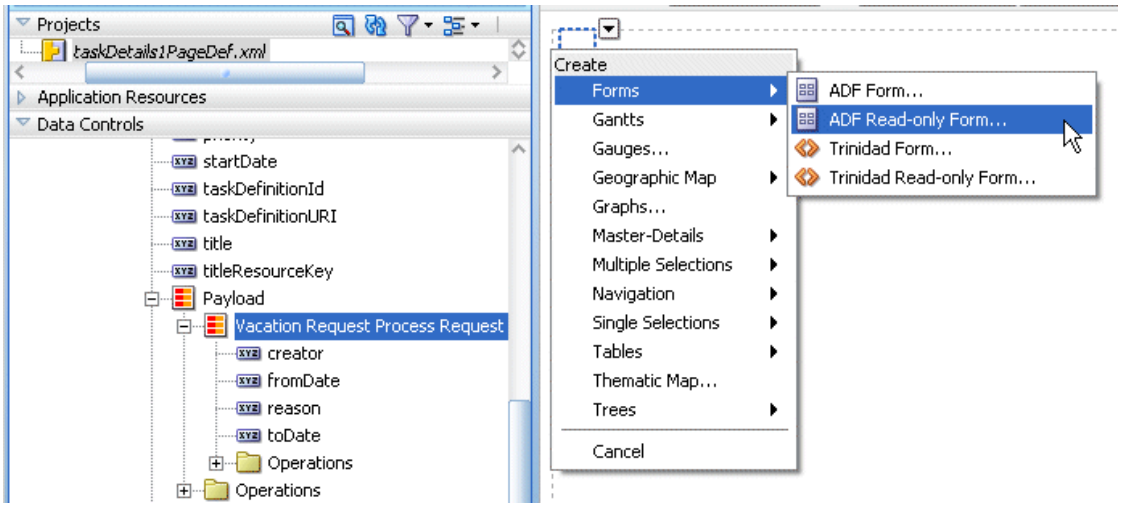
In addition to adding the payload, you can create task form regions. See Step 1 in How To Create Task Form Regions Using Individual Drop Handlers.

To add the payload to the task form:

1. From the **Components window**, 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 30-28.

Figure 30-28 Adding ADF Read-Only Fields to the Task Form Payload Region

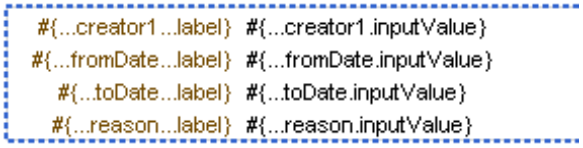


7. In the Edit Form Fields dialog, accept the defaults and click **OK**.

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

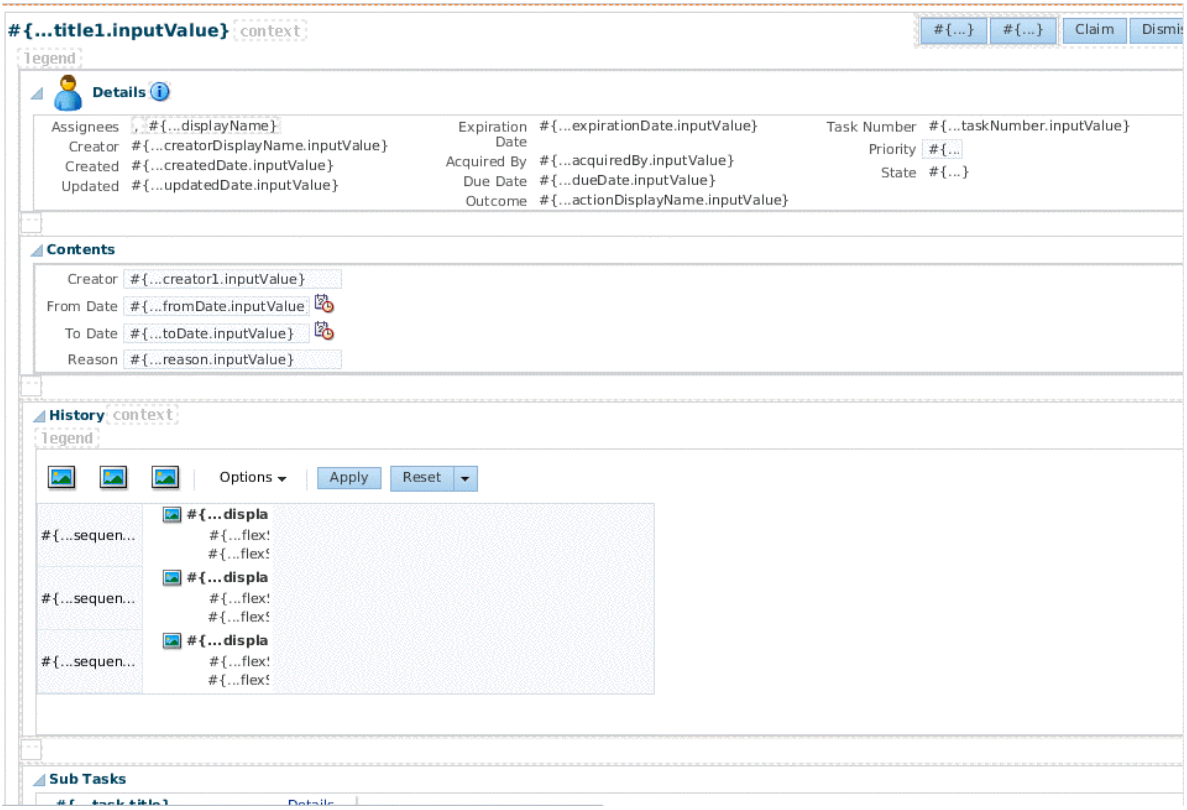
   The payload regions appear, as shown in Figure 30-29.

Figure 30-29 The Payload Region of the Task Form



The task form, shown in Figure 30-30, is complete and ready to be deployed.

Figure 30-30 The Task Form (taskDetails.jspx)



What Happens When You Create a Task Form

The form you designed is saved in the `.jspx` file at

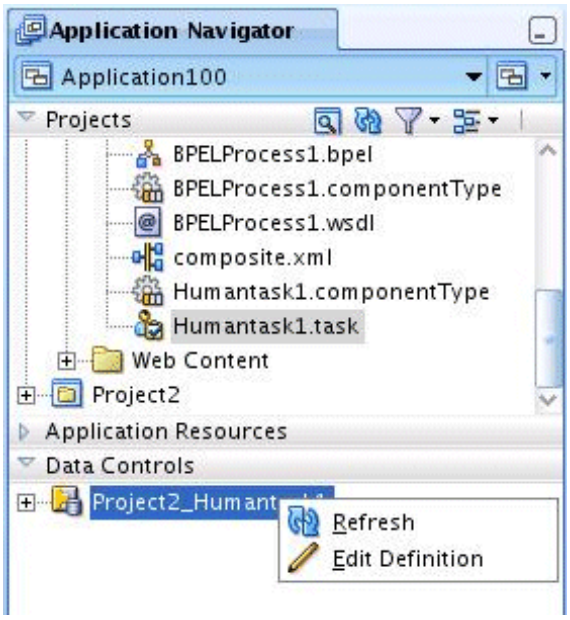
JDev_Oracle_Home`\mywork\`*task_form_application_name*`\`*project_name*`\public_html`

The task form is ready to be deployed. See Deploying a Composite Application with a Task Flow.

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 30-31 shows the **Refresh** option.

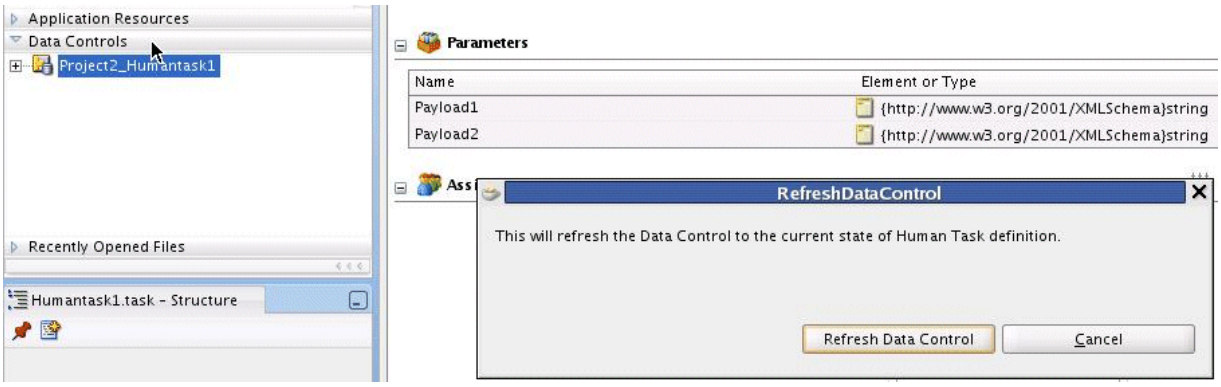
Figure 30-31 Refreshing Data Controls



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 30-32.

Figure 30-32 The Refresh Data Control Button



Securing the Task Flow Application

You can use any container-based security for securing the task flow. See 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.

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.

Notifications from Human Workflow to review notification settings as part of a human task definition (`.task` file).

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 available with the Oracle SOA Suite samples.
- 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.)

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:

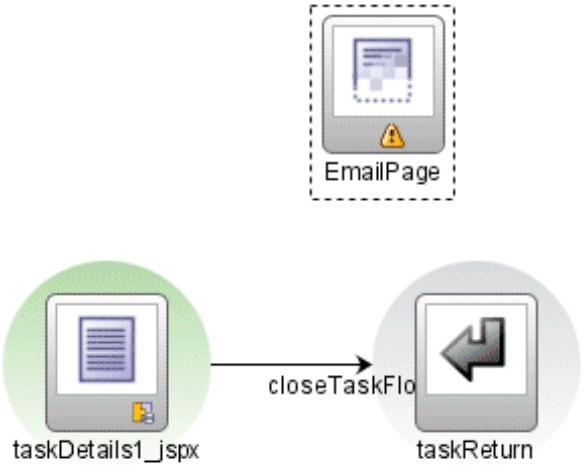
1. In the Applications window, expand the task flow project and double-click *project_name* **_TaskFlow.xml**.

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

2. From the **Components window**, 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 30-33 shows an example using the name **EmailPage**.

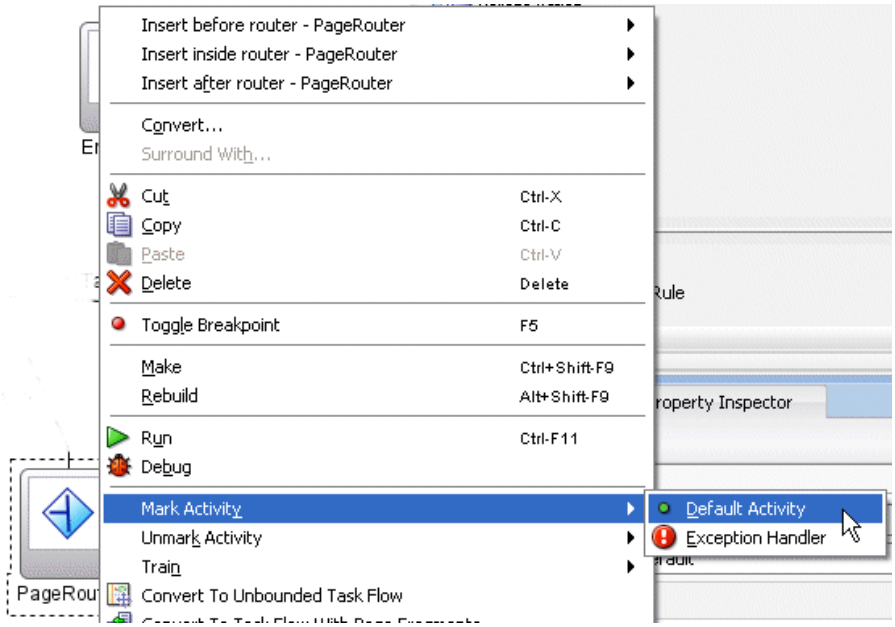
Figure 30-33 Creating the Email Page



4. From the **Components window**, drag **Router** into the designer.
5. Click **router1** below the icon and enter a router name.

   Figure 30-35 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 30-34.

Figure 30-34 Marking the Router as the Default Activity

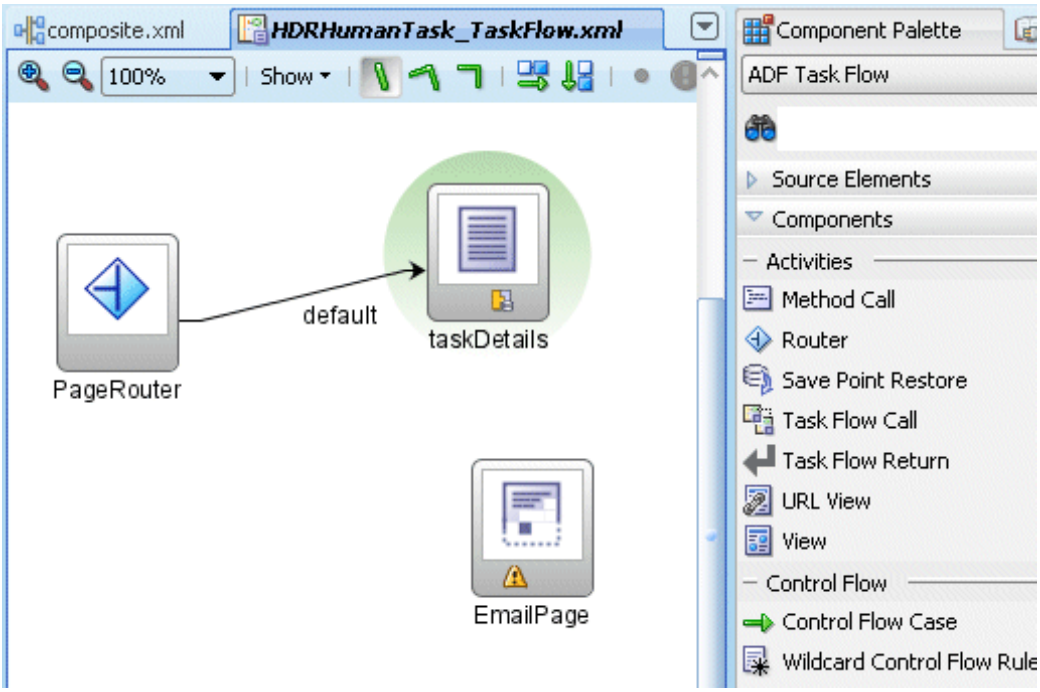


7. Click the **router - *router_name* - Property Inspector** tab.

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 **Components window**, 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 30-35.

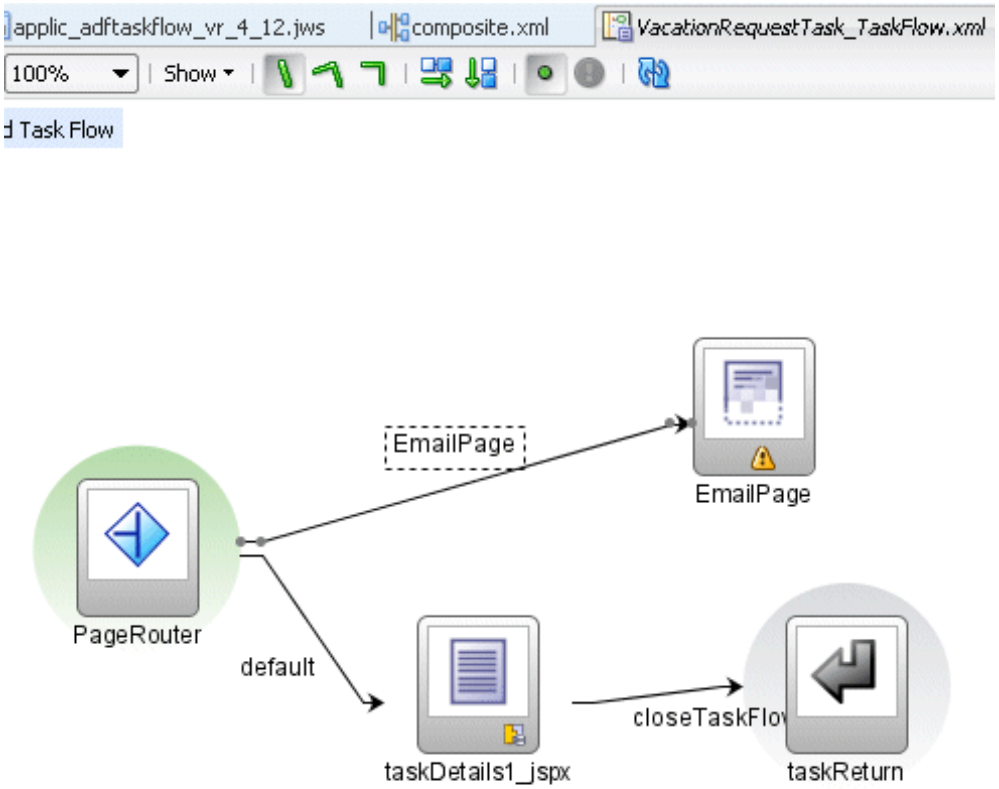
Figure 30-35 Connecting the Control Flow



13. In the **Components window**, 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 30-36 shows the completed control flow.

Figure 30-36 Completed Control Flow for an Email Notification



To continue creating the email notification page, see Step 1 in Creating an Email Notification Page.

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 *Developing Web User Interfaces with Oracle ADF Faces* for design information.

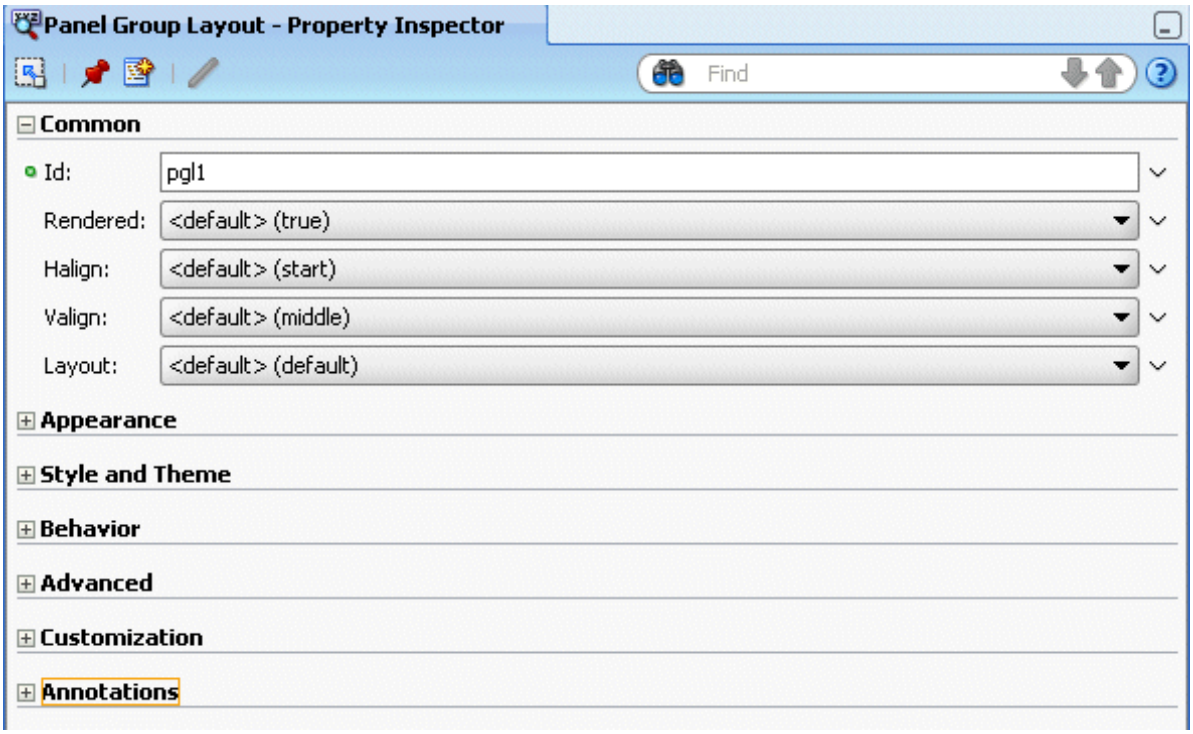
To create an email notification page:

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 **Components window**, 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 30-37 shows the layout fields available when **Panel Group Layout** is selected.

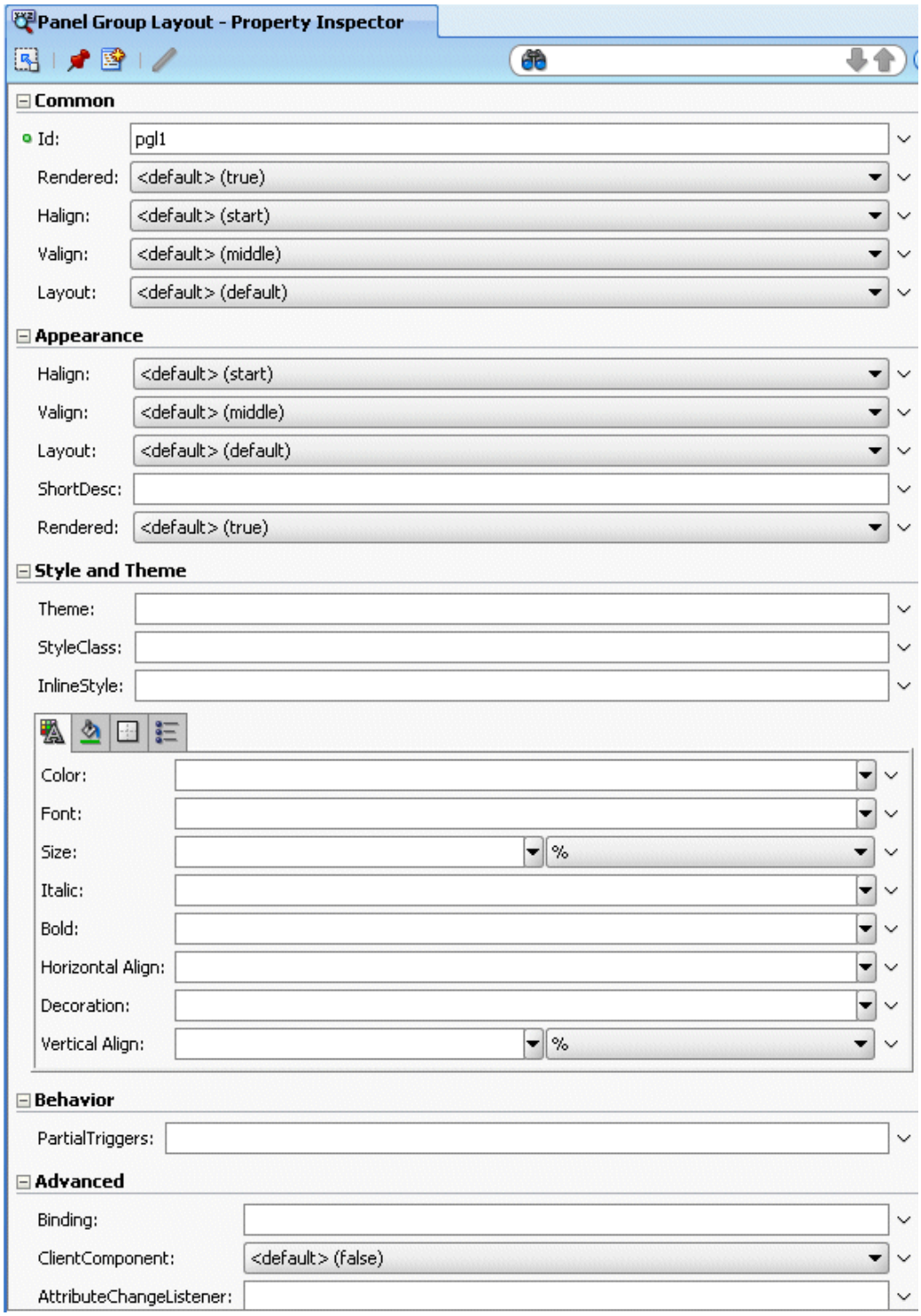
Figure 30-37 Specifying a Layout



See *Developing Web User Interfaces with Oracle ADF Faces* 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 30-38.

Figure 30-38 Specifying a Layout: More Details



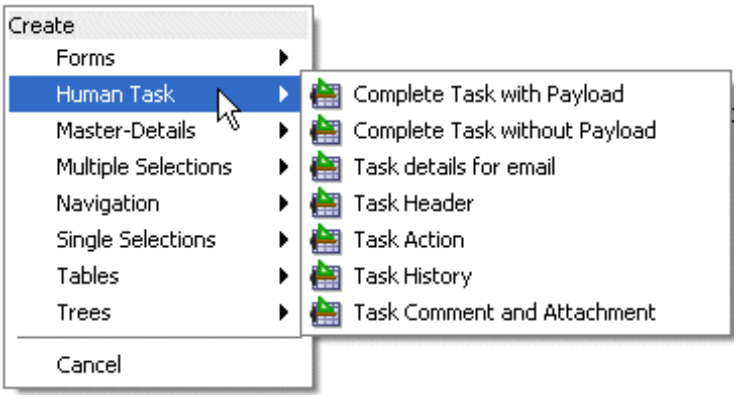
For more information, see "How to Set Component Attributes" in *Developing Web User Interfaces with Oracle ADF Faces*.

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 30-39.

Figure 30-39 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.

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.

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 How To Create an Email Notification, for more information.

Security issues can also prevent the form from being rendered correctly. See Securing the Task Flow Application, for more information.

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 How To Deploy a Composite Application with a Task Flow. See Deploying SOA Composite Applications for more information.

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**.

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.

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 an application and project as a container for the task flow. After you deploy the SOA composite application, deploy the task flow application.

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.

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 must 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.

The following example shows a 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.example.com:8001</serverURL>

<initialContextFactory>weblogic.jndi.WLInitialContextFactory</initialContextFactory>

<participateInClientTransaction>false</participateInClientTransaction>
        </remoteClient>
        <soapClient>

<rootEndPointURL>http://my_host.us.example.com:8001</rootEndPointURL>
        </soapClient>
    </server>
</workflowServicesClientConfiguration>
```

Configuring Unique Cookie Context Paths for the Session Tracking Cookies

Before deploying the task form to a non-SOA Oracle Weblogic Server, ensure 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 such as 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
```

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:

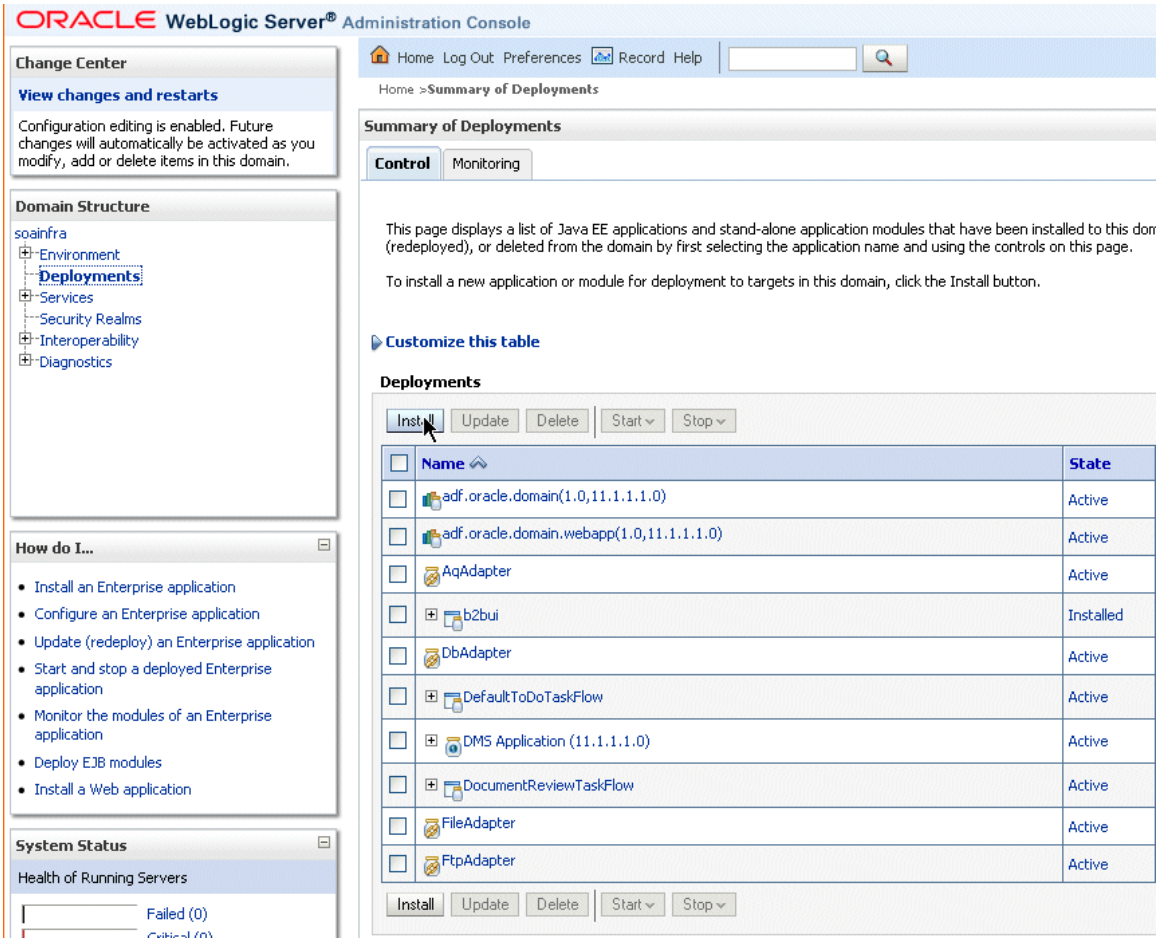
1. Go to Oracle WebLogic Server Administration Console at

   `http://`*`remote_hostname:remote_portnumber`*`/console`

2. In the **Domain Structure** area, click **Deployments**.

3. Click **Install**, as shown in Figure 30-40.

Figure 30-40 Oracle WebLogic Server Administration Console: List of Deployments

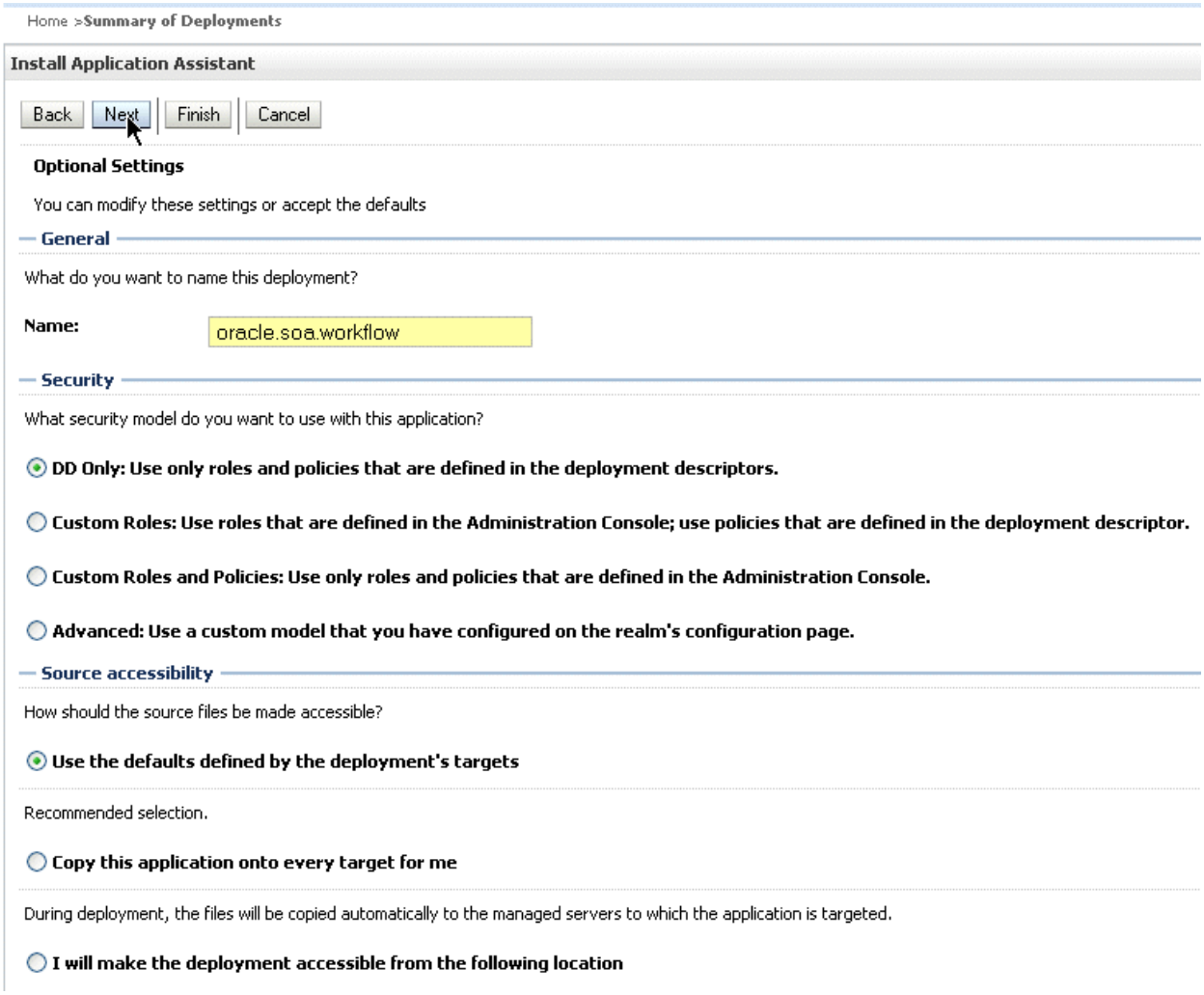


4. In the **Path** field, provide the following path and click **Next**.

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

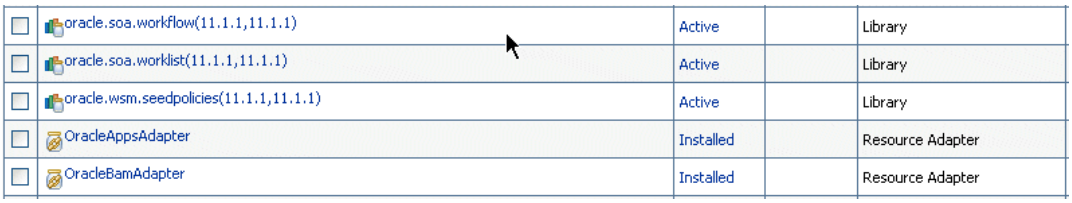
5. Keep the same name for the deployment and click **Next**, as shown in Figure 30-41.

Figure 30-41 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 30-42.

Figure 30-42 Oracle WebLogic Server Administration Console: The oracle.soa.workflow Active State



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

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 30-43, and click **Next**.

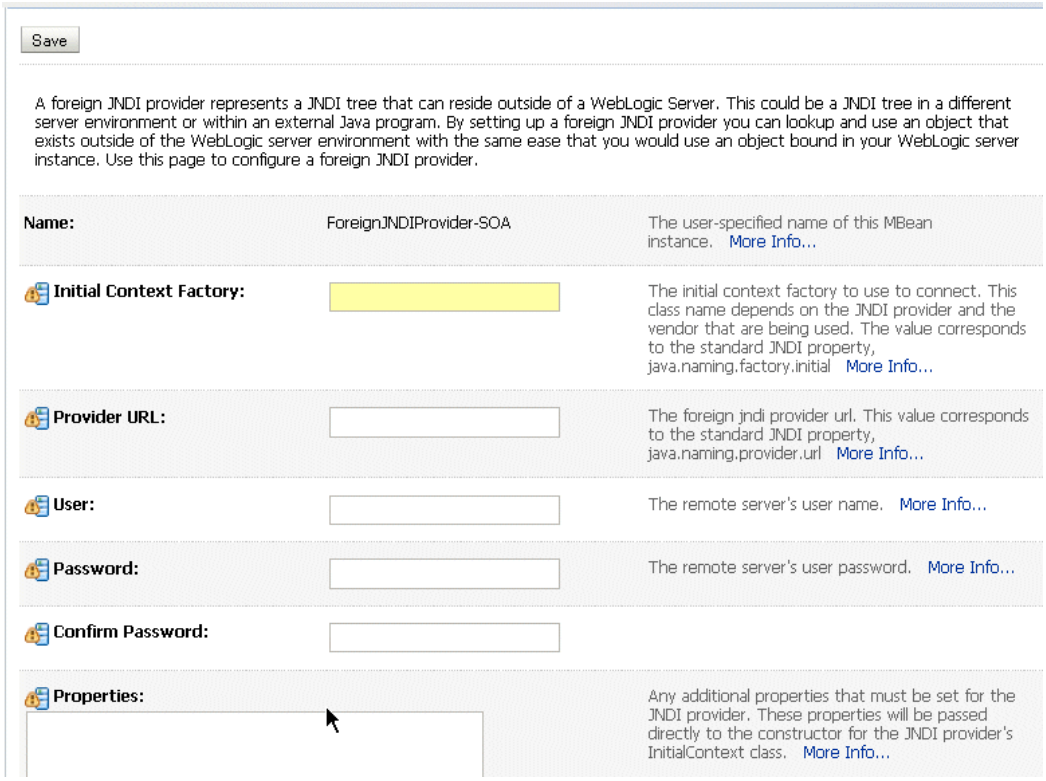
Figure 30-43 Creating a Foreign JNDI Provider

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 30-44 shows the page where you enter this information.

Figure 30-44 Defining the Foreign JNDI Provider



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

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

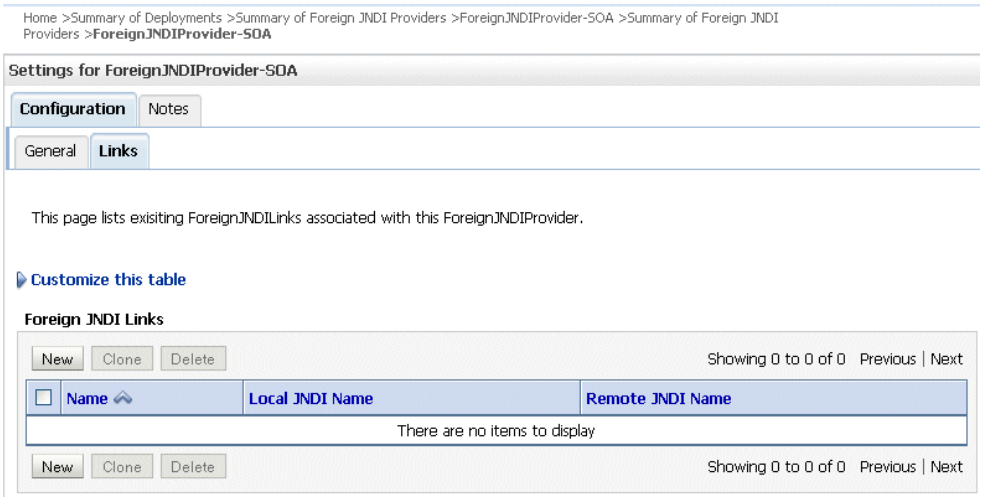
Use Oracle WebLogic Server Administration Console to complete this portion of the task.

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 30-45 shows the **Links** tab.

Figure 30-45 Defining the Foreign JNDI Provider Links: The Links Tab

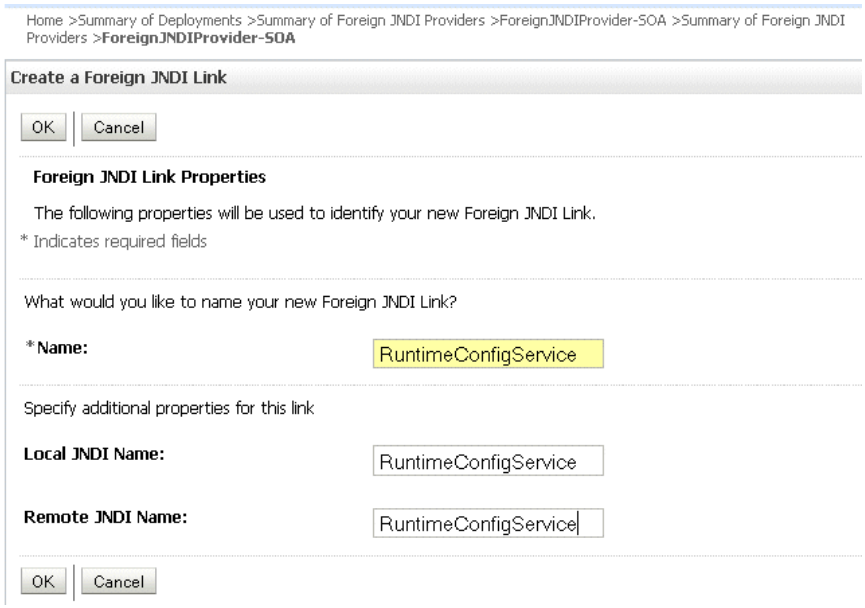


5. Do the following and click **OK**.

   - For **Name**, enter `RuntimeConfigService`.
   - For **Local JNDI Name**, enter `RuntimeConfigService`.
   - For **Remote JNDI Name**, enter `RuntimeConfigService`.

   Figure 30-46 shows where you do this.

Figure 30-46 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 Including a Grant for bpm-services.jar, to continue.

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 Deploying the Application, to continue.

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 Including a Grant for bpm-services.jar.. See Deploying SOA Composite Applications for information on deploying applications.

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 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 *Administering Oracle SOA Suite and Oracle Business Process Management Suite* for more information. If the task flow is configured for HTTPS access, you may need to do additional settings in Enterprise Manager.

Note:

For the task form association to happen automatically, the SOA server must be running. If the association does not happen, then you receive the message `Task details not found for this task` when you try to access the task form for that task in Worklist Application or Oracle Business Process Management Workspace. In this case, you can either restart the application or go to Oracle Enterprise Manager and register the task form URL manually.

See Using for information on how to act on tasks.

Note:

- 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.

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

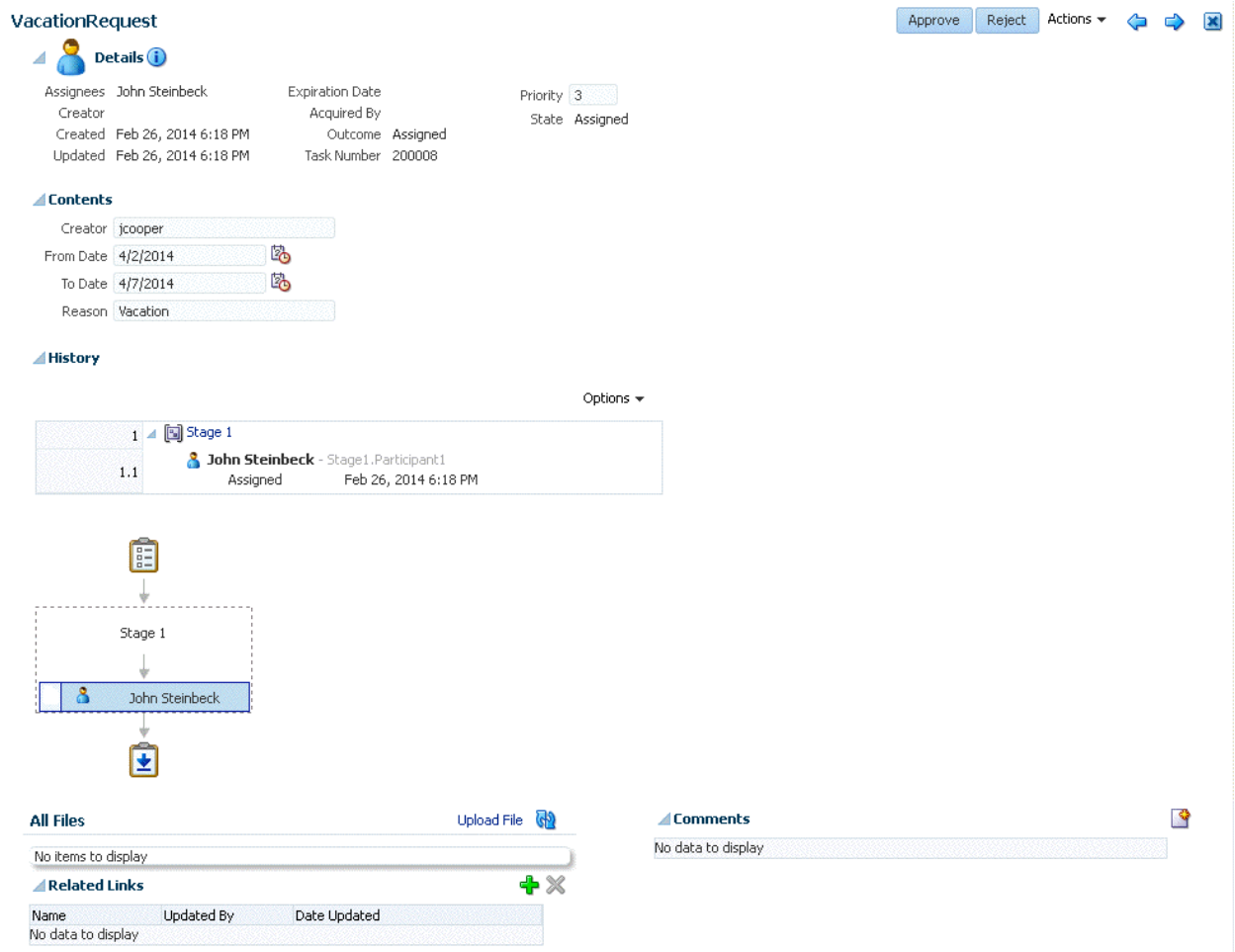
Manager Fusion Middleware Control to undeploy the task flow application from the application home page.

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 30-47 shows how the task form for the help desk request example is displayed in the Worklist Application task details page.

Figure 30-47 Worklist Task Details Page

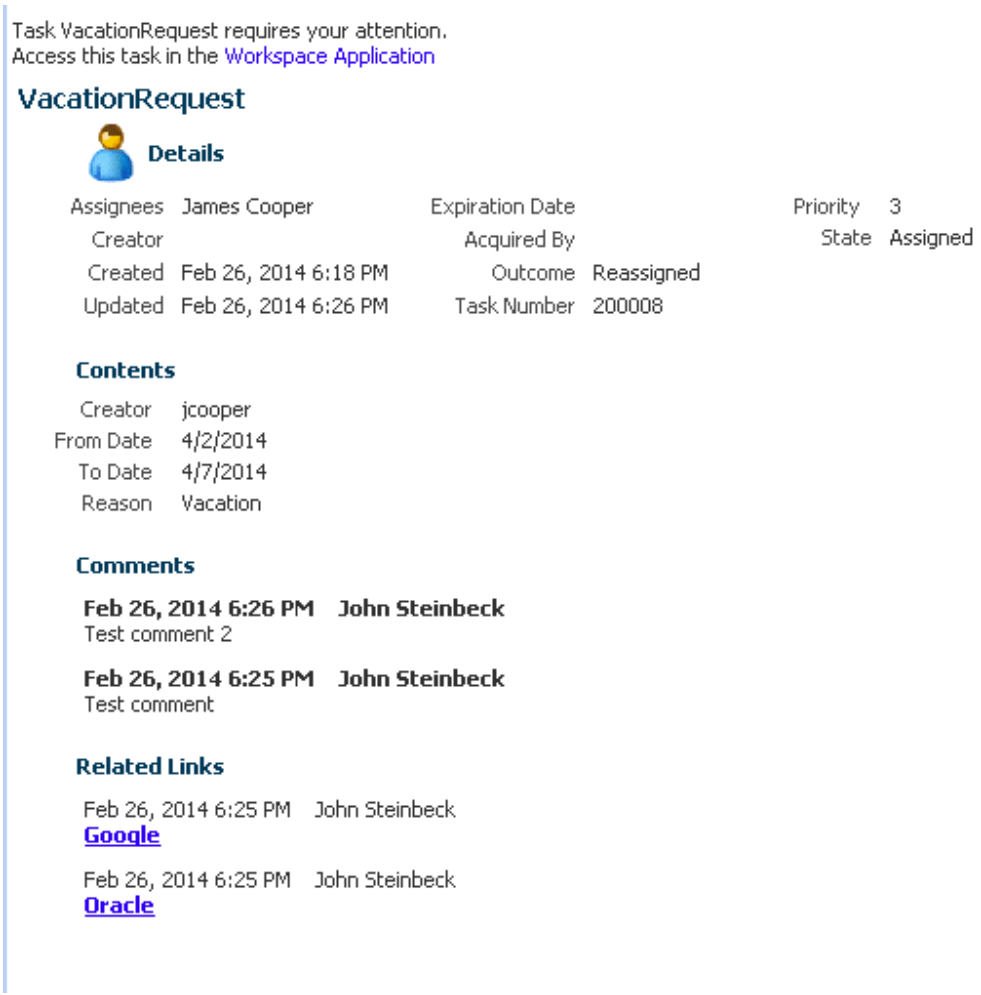


The task form is available in Oracle BPM Worklist after you log in. See How To Log In to the Worklist for instructions.

Displaying a Task in an Email Notification

Figure 30-48 shows how an email task notification appears in email.

Figure 30-48 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.

Changing the Text for the Worklist Application in Task Notifications

By default, the text in a task notification refers to "Worklist Application," but you can change that text. To change it, 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.

   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:///...`.

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 Config** > **Task** tab.

   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
   ```

For information about showing or hiding the URL of the Worklist Application, see How to Display the URL in Notifications.

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 both the payload elements and.the outcomes must be identical.

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.

How to Reuse the Task Flow Application with Different Actions

You can reuse a single task flow that has different actions for different tasks. To do this:

1. Define all actions in the task that you use to generate the taskflow.
2. In any given task, disable the actions that you do not want to include.

31

Human Workflow Tutorial

This chapter describes how to design your first workflow from start to finish.

This chapter includes the following sections:

Introduction to the Human Workflow Tutorial

The application developed in this tutorial is based on the following use-case:

- an employee submits a vacation request
- the manager approves or rejects the vacation request
- the employee receives a notification that approves or rejects their request

This tutorial describes how to create a new application and SOA project and how to design a human task to send a vacation request to a manager for approval or rejection.

The resulting SOA composite application contains the following components:

- A BPEL process
- A human task, for approving a vacation request submitted by an employee

It also describes how to create an Oracle ADF-based task form that enables the end user to act upon the vacation request once the application is deployed and running. To

create an Oracle ADF-based task form you must create a new application and a new project.

This tutorial guides you through the following tasks:

- Using the
- Using the 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

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 and Oracle BPEL Designer, the environment for designing and deploying BPEL processes, is also assumed.

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 Additional Tutorials for information on downloading this and other tutorials.

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. From the **File** main menu, select **New > Applications > SOA Application**.

   Click **OK**.

2. In the **Application Name** field, enter `VacationRequest`, and click **Next**.

3. In the **Project Name** field, enter `VacationRequest`, and click **Next**.

4. In the **Composite Template** list, select **Composite with BPEL Process**, and click **Finish**.

   The Create BPEL Process dialog appears.

5. In the **Name** field, enter `VacationRequestProcess`.

   Go to the bottom of the Create BPEL Process dialog.

6. To the right of the **Input** field, click the **Search** icon.

   The Type Chooser dialog appears.

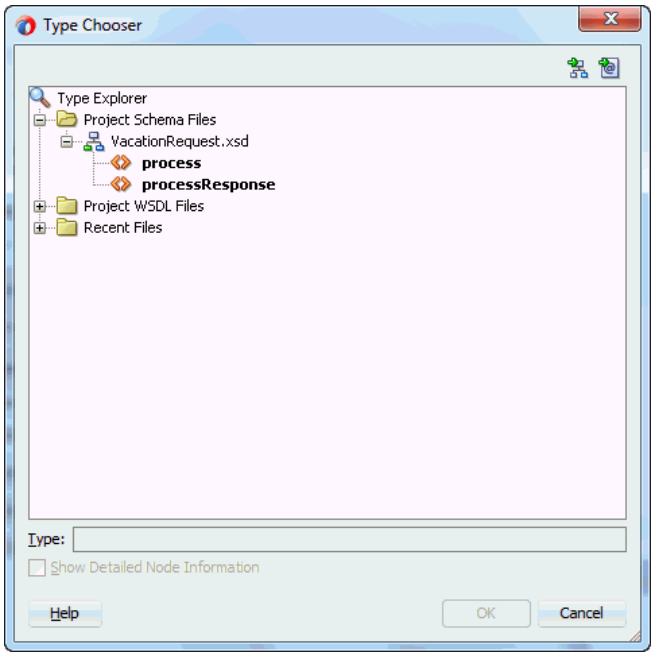
7. In the upper right corner, click the **Import Schema File** icon.

   The Import Schema File dialog appears.

8. Browse for and select the **VacationRequest.xsd** file you created in Prerequisites.

   Click **OK** until you are returned to the Type Chooser dialog, as shown in Figure 31-1.

Figure 31-1 Type Chooser Dialog with the Request and Response Elements



9. Select the input element **VacationRequestProcessRequest**, and click **OK**.

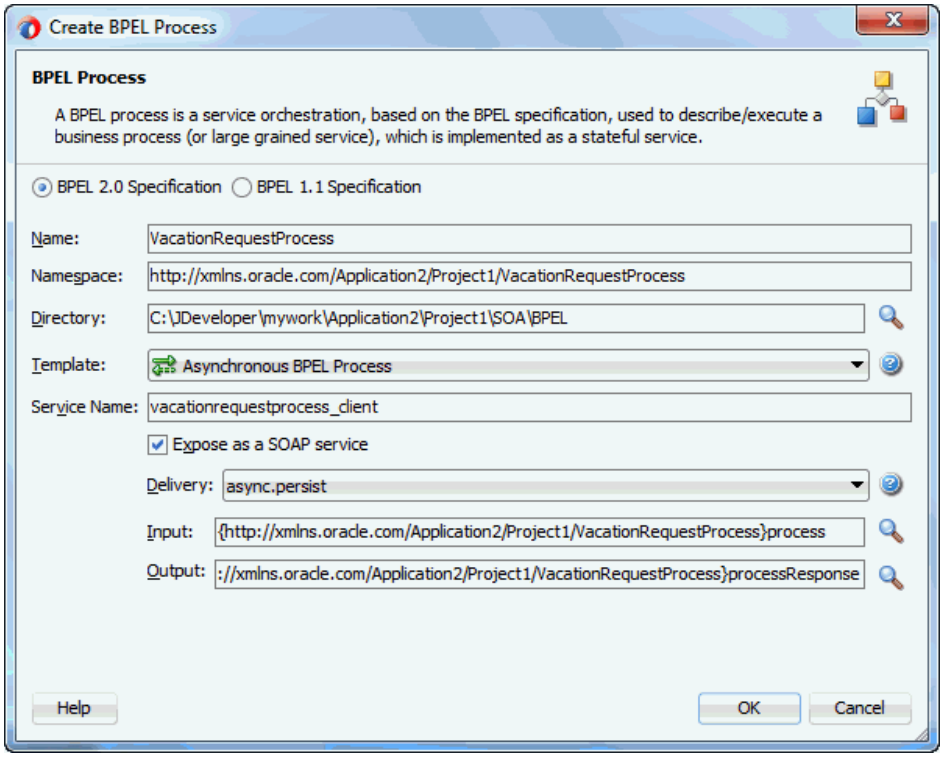
   You are returned to the Create BPEL Process dialog.

10. To the right of the **Output** field, click the **Search** icon.

11. Select the output element **VacationRequestProcessResponse**, and click **OK**.

    You are returned to the Create BPEL Process dialog, as shown in Figure 31-2.

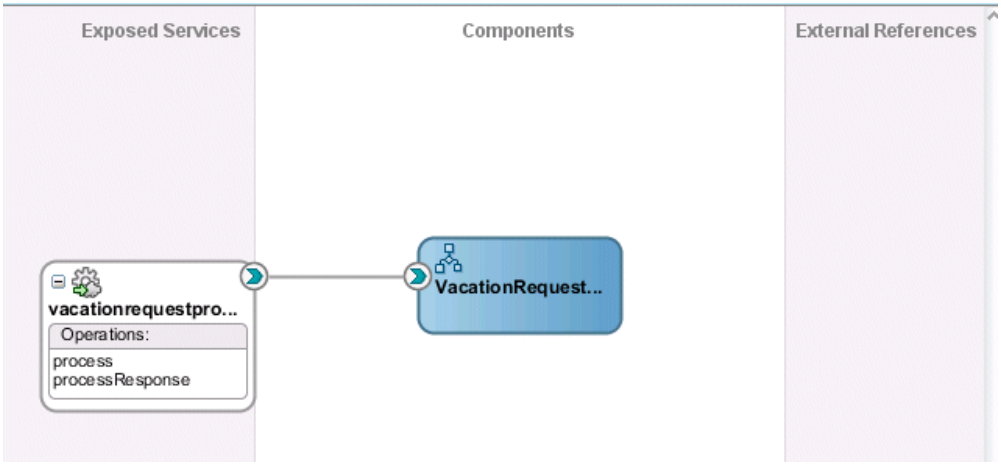
Figure 31-2 BPEL Process Dialog



12. 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 31-3. 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 31-3 BPEL Process in SOA Composite Editor



For more information about service components and the SOA Composite Editor, see Getting Started with Developing SOA Composite Applications.

Creating 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 Components window, drag a **Human Task** into the .

   The Create Human Task dialog appears.

2. Enter the details described in Table 31-1.

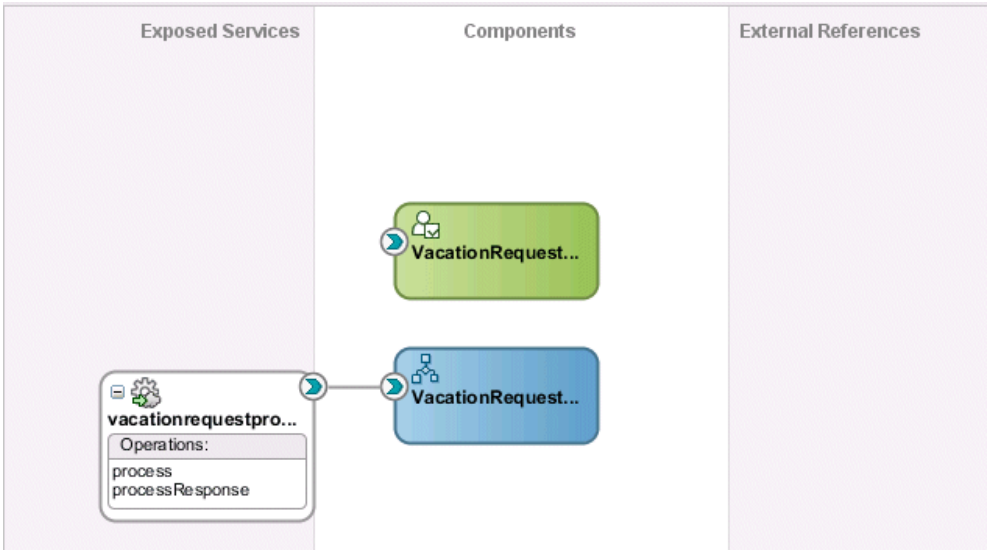
Table 31-1 Create Human Task Dialog Fields and Values

Field	Value
Name	Enter `VacationRequestTask`.
Namespace	Accept the default value.
Create Composite Service with SOAP Bindings	Do *not* select the check box. Instead, you create a human task that you later associate with the BPEL process you created in 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 above the BPEL process, as shown in Figure 31-4.

Figure 31-4 Human Task Icon in



4. Double-click the **Human Task** icon.

   The Human Task Editor appears. You are now ready to begin design of your human task.

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 and click the **Add** icon to specify the task payload
4. Select **Add string parameter**.

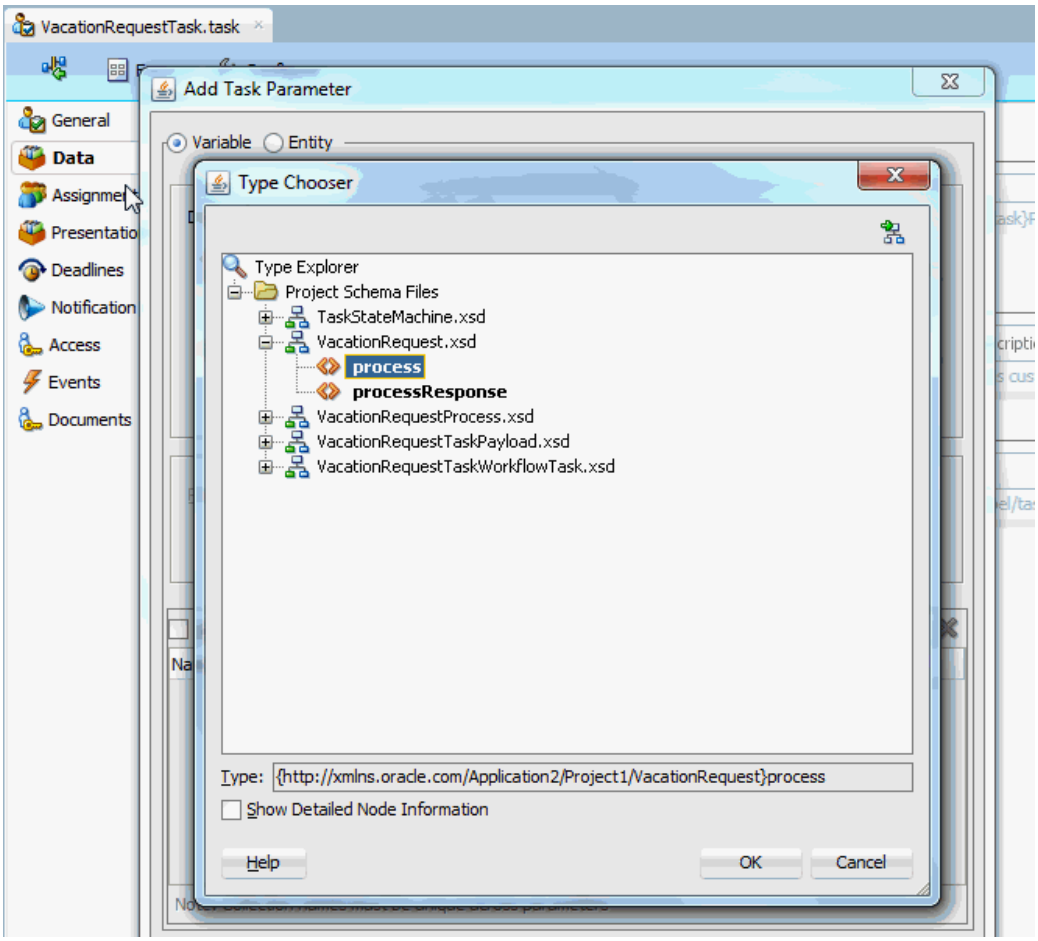
   The Add Task Parameter dialog is displayed. You now create parameters to represent the elements in your XSD file. This makes the payload data available to the workflow task.

5. Select **Element**. To the right of the **Element** field, click the **Search** icon.

   The Type Chooser dialog appears.

6. Expand and select **Project Schema Files** > **VacationRequest.xsd** > **process**, and click **OK**. Figure 31-5 provides details.

Figure 31-5 Type Chooser Dialog



Ensure that the **Editable via worklist** check box is selected. This provides you with the option to modify this parameter during runtime from Oracle BPM Worklist.

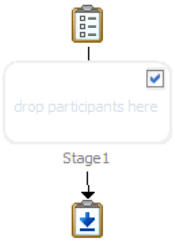
Click **OK** on the Add Task Parameter dialog.

7. Click the **Assignment** tab on the left side of the editor.

8. From the Participants section from the Components window, grab a Single Participant type and drop it in the <Drop participant here> box, as shown in Figure 31-6. You select this type because a single assignee, the manager, acts on the vacation request task.

   Oracle SOA Suite provides several out-of-the-box patterns known as participant types for addressing specific business needs. For more information, see Task Assignment and Routing.

Figure 31-6 Assignment and Routing Policy



9. Double-click the participant you added.

   The Edit Participant Type dialog box opens.

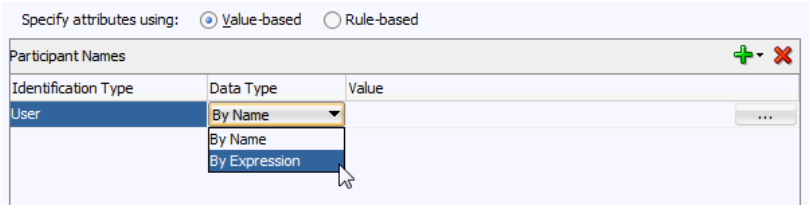
10. In the **Participant Names** table, click the **Add** icon, and select **Add User**.

    This participant type acts alone on the task.

11. Click the **Data Type** column, and select **By Expression** from the list that is displayed. Figure 31-7 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 Creating an Application and a Project with a BPEL Process). The task is automatically routed to the employee's manager.

Figure 31-7 Selection of By Expression from the Data Type Column



12. In the **Value** column, click the **Browse** icon (the dots) to invoke the Expression Builder dialog.

13. In the dropdown list in the **Functions** section, select **Identity Service Functions**.

14. Select **getManager**. This function gets the manager of the user who created the vacation request task.

15. Above the **Functions** section, click **Insert into Expression**. Place the cursor between the parentheses of the function.

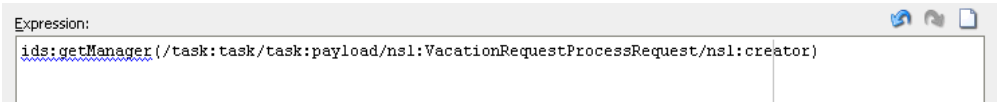
16. In the **Schema** section, expand **task:task** > **task:payload** > **ns1:VacationRequestProcessRequest** > **ns1:creator**.

    where **ns1** is the namespace for this example; your namespace may be different.

17. Click **Insert into Expression**.

The Expression Builder dialog displays the XPath expression in the **Expression** section. Figure 31-8 provides details.

Figure 31-8 XPath Expression



18. Click **OK** to exit the Expression Builder dialog. Again, click **OK** to exit the Add Participant Type dialog.
19. From the **File** menu, select **Save All**.

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 Creating an Application and a Project with a BPEL Process.

To associate the human task and BPEL process service component:

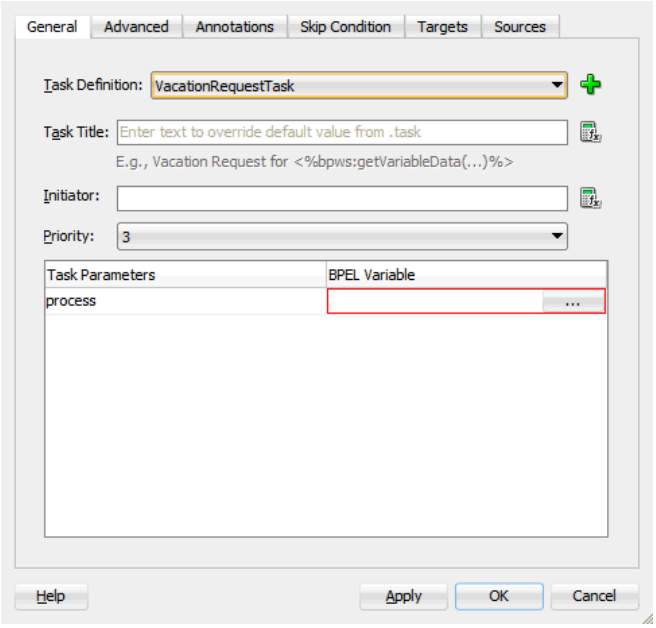
1. In the Applications window, double-click **composite.xml**.
2. Double-click the **VacationRequestProcess** BPEL process service component in the .

   The BPEL process displays in Oracle BPEL Designer.
3. In the Components window, expand **SOA Components**.
4. Drag a **Human Task** beneath the **receiveInput** receive activity. Double-click the activity.

   The Human Task dialog appears.
5. From the **Task Definition** list, select the **VacationRequestTask** task you created (if it is not currently displaying).

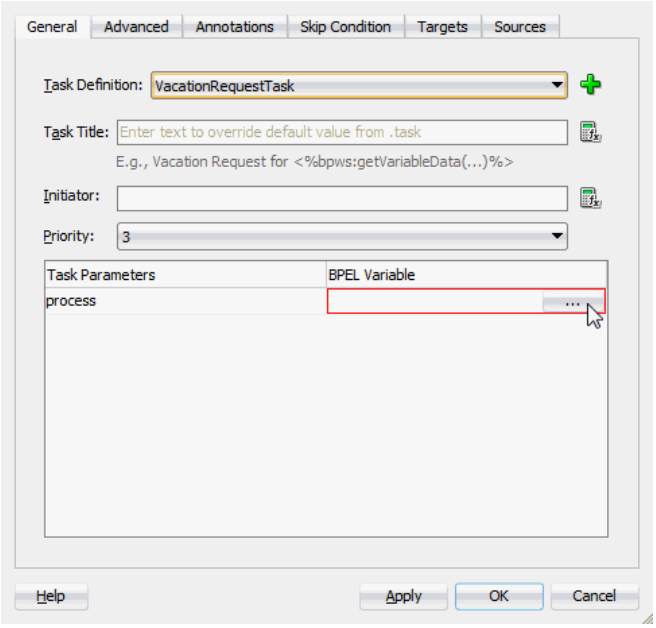
   The dialog refreshes as shown in Figure 31-9 to display additional fields.

Figure 31-9 Human Task Dialog



6. In the **BPEL Variable** column, click the **Browse** icon (dots) shown in Figure 31-10.

Figure 31-10 BPEL Variable Entry

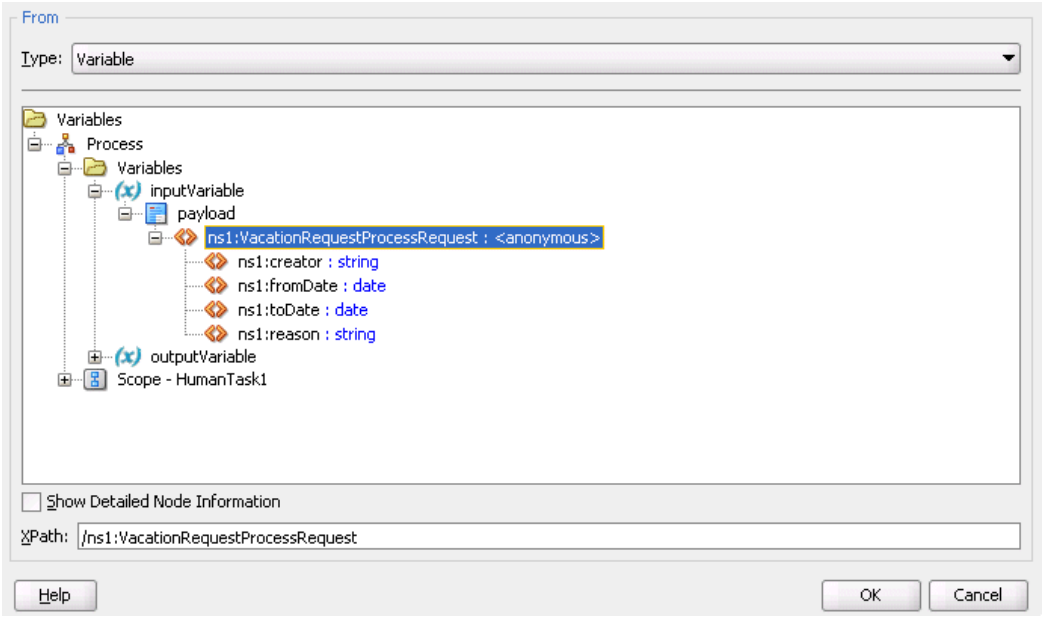


The Task Parameters dialog appears.

7. From the **Type** list, select **Variable**.

8. Expand **Process** > **Variables** > **inputVariable** > **payload** > **ns1:VacationRequestProcessRequest**. Figure 31-11 provides details.

Figure 31-11 Variable Selection

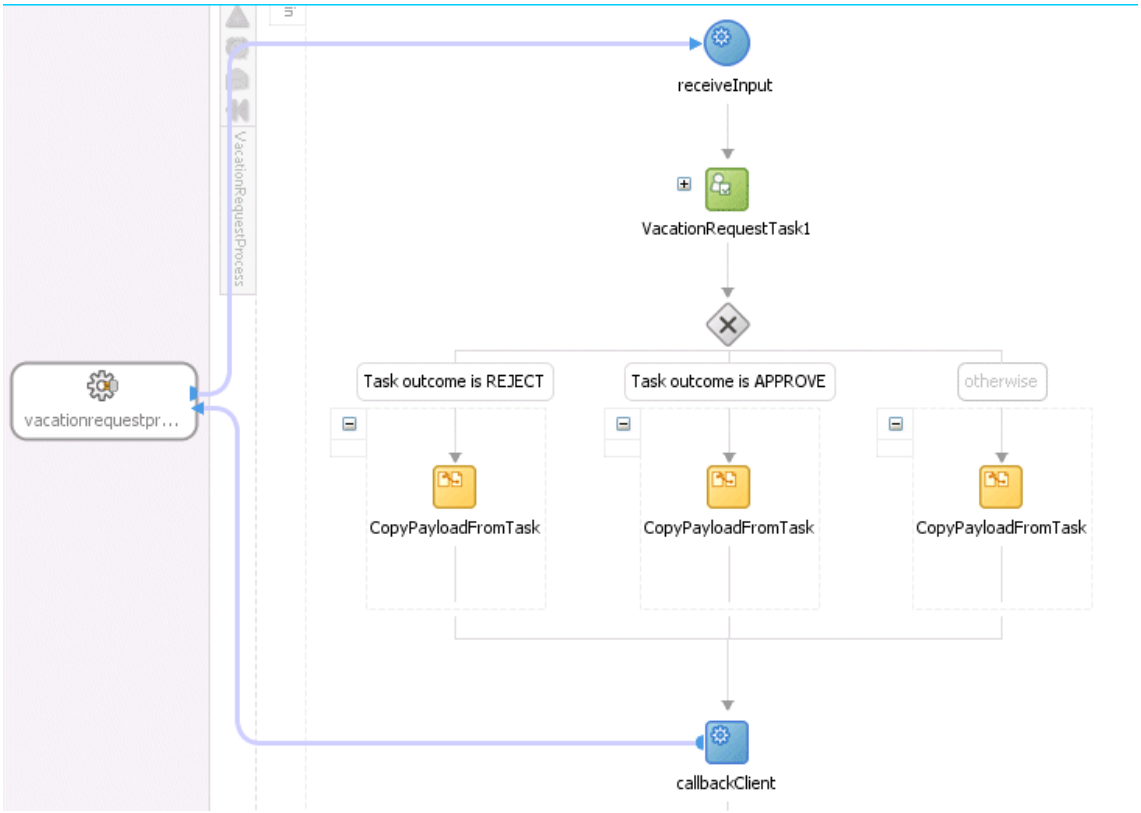


Click **OK**.

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

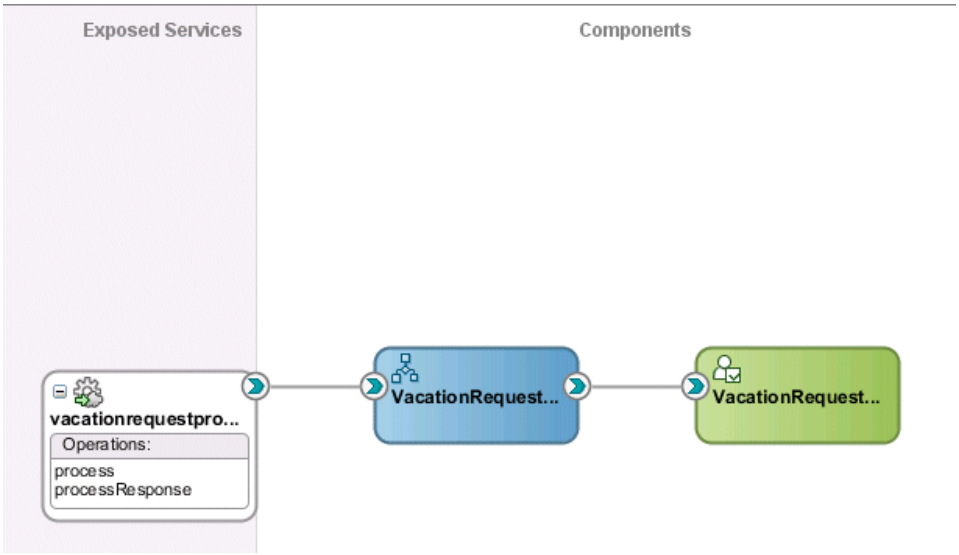
   The human task activity appears as shown in Figure 31-12.

Figure 31-12 Human Task and Partner Links in Oracle BPEL Designer



10. Return to the SOA Composite Editor and note that the BPEL process and human task service components have been automatically connected. Figure 31-13 provides details. From the **File** menu, select **Save All**.

Figure 31-13 SOA Composite Editor



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.

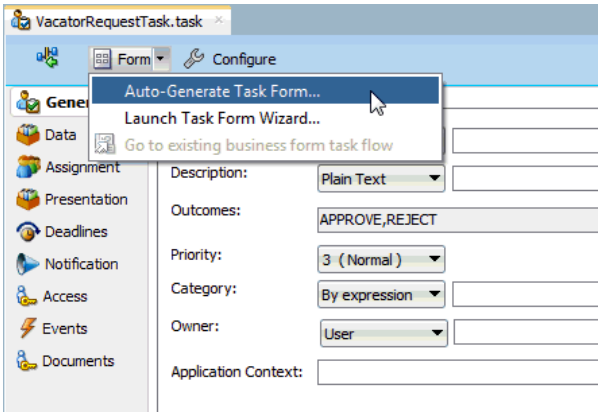
To create a task form project:

1. Double-click the **VacationRequestTask** human task.

   The Human Task Editor is displayed.

2. From the **Form** menu at the top, select **Auto-Generate Task Form**. Figure 31-14 provides details.

Figure 31-14 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**.

Deploying the Task Form

To deploy the task form:

1. In the Applications window, 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 Deploying SOA Composite Applications in .
3. Return to Oracle BPM Worklist.
4. Note that the task form now appears at the bottom of Oracle BPM Worklist.

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 for Oracle Identity and Access Management*.

To create an application server connection

1. From the **File** main menu, select **New** > **Connections** > **Application Server Connection**.

   Click **OK**.
2. In the **Connection Name** field, enter a connection name.
3. From the **Connection Type** list, select **WebLogic 10.3**.

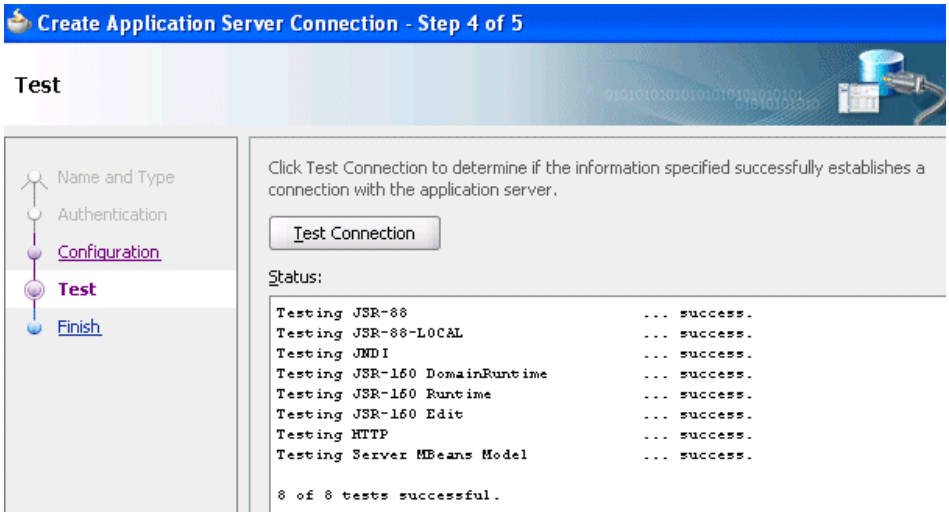
   Click **Next**.
4. In the **Username** field, enter `weblogic`.
5. In the **Password** field, enter the password for connecting to the application server.

   Click **Next**.
6. Enter the hostname for the application server that is configured with the SOA Infrastructure.
7. In the **Weblogic Domain** field, enter the Oracle WebLogic Server domain.

   Click **Next**.
8. Click **Test Connection**.

   If successful, the message shown in Figure 31-15 is displayed.

Figure 31-15 Connection Success



9. Click **Finish**.
10. From the **File** menu, select **Save All**.

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 Applications window, 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 Deploying SOA Composite Applications in .

Initiating the Process Instance

See *Administering Oracle SOA Suite and Oracle Business Process Management Suite* for instructions on accessing the Test Web Service page for initiating the process instance.

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.

Additional Tutorials

In addition to the vacation request use case, other tutorials are available from the Oracle SOA Suite samples.

Table 31-2 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 31-2 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
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

Table 31-2 (Cont.) End-to-End Examples

Sample	Description	Name
Java Samples	Provides an assortment of samples that use Java to interact with human workflow.	workflow-118-JavaSamples

32

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, Building a Custom Worklist Client.

For information about troubleshooting human workflow issues, see section "Human Workflow Troubleshooting" of *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

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.

Oracle BPM Worklist provides different functionality based on the user profile. Standard user profiles include task assignee, supervisor, process owner, reviewer, and administrator. For example, worklist users can update payloads or business data,

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.

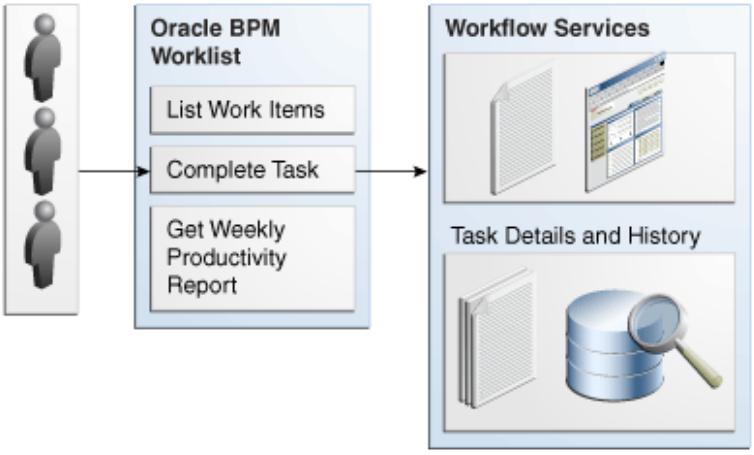
Users can filter their tasks by creating views or saved searches.

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 32-1 shows an illustration of Oracle BPM Worklist.

Figure 32-1 Oracle BPM Worklist—Access Tasks, Forms, Attachments, and Reports



The worklist is the list of tasks. A task form displays and updates the task details. You can create a task form using ADF task flows in Oracle JDeveloper. See Designing Task Forms for Human Tasks for more information.

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.

Logging In to Oracle BPM Worklist

Table 32-1 lists the different types of users recognized by Oracle BPM Worklist, based on the privileges assigned to the user.

Table 32-1 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.
Assignee	Acts on tasks assigned to him, in addition to his own end-user access
Reviewer	Acts on tasks assigned for review, in addition to his own end-user access

Note:

Multiple authentication providers (for example, SSO and forms) are not supported.

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 *Installing and Configuring Oracle SOA Suite and Business Process Management* 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 *Administering Oracle SOA Suite and Oracle Business Process Management Suite* for the organizational hierarchy of the demo user community used in examples throughout this chapter.

3. Click **Login**.

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:

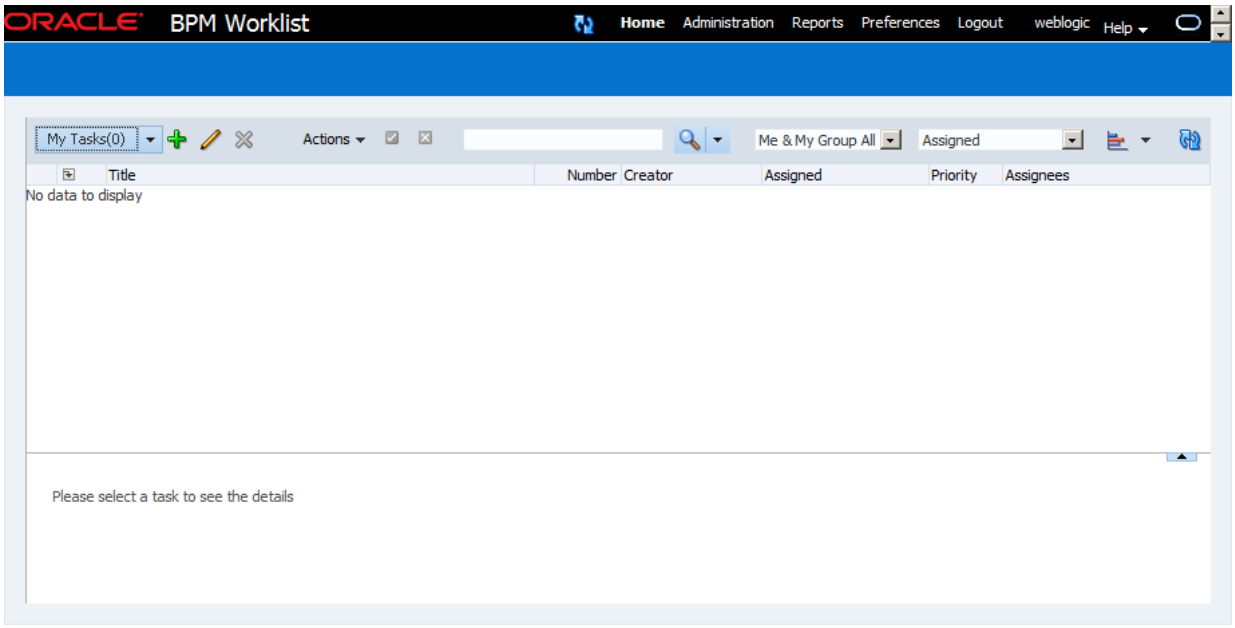
```
dn: cn=Administrators,cn=Groups,dc=us,dc=oracle,dc=com
objectclass: groupOfUniqueNames
objectclass: orclGroup
objectclass: top
owner: cn=orcladmin,cn=Users,dc=us,dc=oracle,dc=com
uniquemember: cn=weblogic,cn=Users,dc=us,dc=oracle,dc=com
```

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 How to Specify the Login Page Realm Label, for information on how administrators can set a preference to change the realm label displayed in the interface, or specify an alternative location for the source of the login page image.

Figure 32-2 shows an example of the **Home** page.

Figure 32-2 Oracle BPM Worklist—The Home (Task List) Page



This page lists all the tasks and work items assigned to you, depending on your role. For example, all users can access the My Tasks and Initiated Tasks pages. Only supervisors can access the My Staff page, and only Process Workspace administrators can access the Administrative Tasks page.

At the far left, as shown in Figure 32-3, is a list of views with **My Tasks** selected. Expand this list to select:

- A particular view showing the number of open tasks for each view. Selecting a particular view refreshes the task count to the latest number.
- A list of applications deployed to Process Workspace
- Any favorite links or applications you may have specified

To keep this list visible while you work on tasks, click **Pin**. Then, to hide it, click **Unpin**

Figure 32-3 Selecting a View

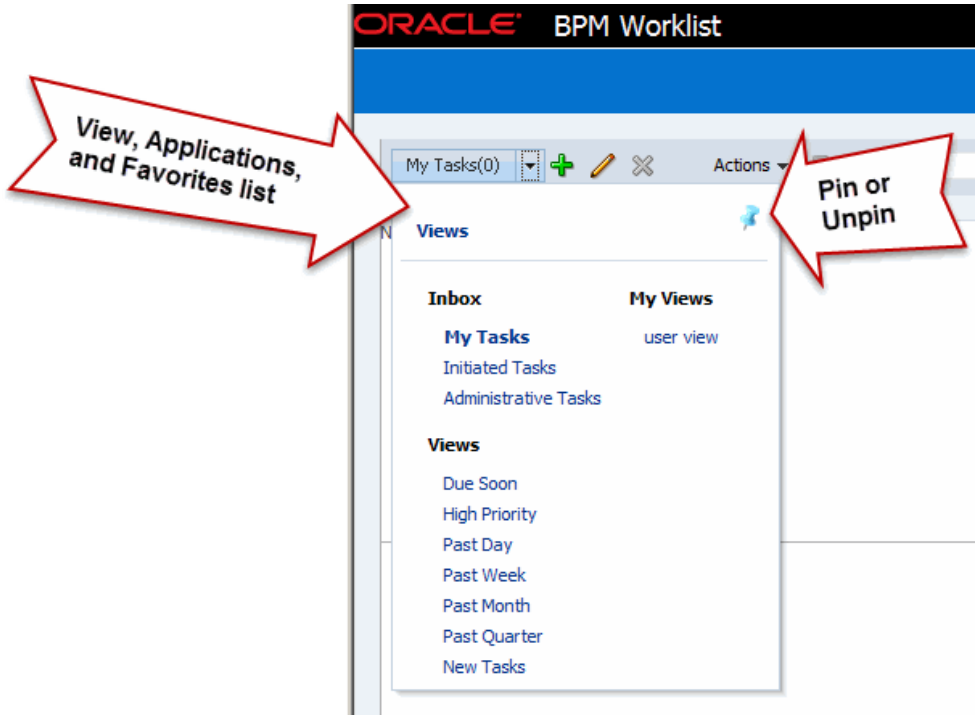


Table 32-2 describes the components of the **Home** (task list) page.

Table 32-2 Components of the Home (Task List) Page

Component	Description
Views list	**Inbox, Standard Views**, **My Views**—See How To Create_ Delete_ and Customize Worklist Views, for more information. The inbox views displayed depend on the role granted to the logged-in user. • Everyone (the user role) sees **My Tasks**, **Initiated Tasks** and **Administrative Tasks**. • Users who are also managers see the **My Tasks**, **Initiated Tasks**, **Administrative Tasks** and **My Staff Tasks** tabs. • Users who are also administrators (the BPMWorkflowAdmin), but not managers, see the **My Tasks**, **Initiated Tasks**, **Administrative Tasks**, **Administration**, and **Evidence Search** tabs. • Users who are managers and administrators see all the tabs— **My Tasks**, **Initiated Tasks**, **My Staff Tasks, Administrative Tasks**, **Administration**, and **Evidence Search**. • Users with the workflow.admin.evidenceStore permission also see the **Evidence Search** tab. See the following for more information: • How To Act on Tasks That Require a Digital Signature, for information about evidence search • How To Manage Other Users' or Groups' Rules (as an Administrator)
Worklist Views	-
Task Status	A bar chart shows the status of tasks in the current view. See How To Customize the Task Status Chart, for more information.

Table 32-2 (Cont.) Components of the Home (Task List) Page

Component	Description
Display Filters	Specify search criteria from the **Assignee** or **State** fields. The category filters that are available depend on which tab is selected. • From the **My Tasks** tab, the **Assignee** filters are **Me**, **My Group**, **Me & My Group**, **Me (Previously)** (tasks worked on previously), and **Me (Review Only)**. From the **Initiated Tasks** tab, the assignee filter is not available. From the **My Staff Tasks** tab, the only assignee filter is **Reportees**. From the **Administrative Tasks** tab, the assignee filter is not available. • The **State** filters include **Any**, **Assigned**, **Completed**, **Suspended**, **Withdrawn**, **Expired**, **Errored**, **Alerted**, **Information Requested**. Use S**earch** to enter a keyword, or use **Advanced Search.** See How To Filter Tasks, for more information.
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, even if the task is an FYI task; one user must claim the task before it can be worked on. 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**—A unique ID number assigned to the task. **Creator**—The user who created the task. **Assigned**—The date that the task was assigned. **Priority**—The priority specified when the human task was created. The highest priority is 1; the lowest is 5.
Task Details	Task details can be viewed in the lower half of the worklist by selecting the task in the Inbox. You can also view them in the same window or a new window by hiding the task details pane in Edit Inbox Settings. After you complete a task: • The Task Details page for the completed task disappears. • The task list refreshes to show only the remaining tasks. • The details of the next open task are shown. See Acting on Tasks: The Task Details Page, for more information.

Figure 32-2 also shows the **Administration**, **Reports**, and **Preferences** links (upper-right corner). Table 32-3 summarizes the **Home**, **Administration**, **Reports**, and **Preferences** pages.

Table 32-3 Worklist Main Pages Summary

Page	Description
Home	As described in Table 32-2, 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.

Table 32-3 (Cont.) Worklist Main Pages Summary

Page	Description
Administration	The following administrative functions are available: • Setting application preferences • Mapping attributes • Searching the evidence store • Administering approval groups • 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 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

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.

Customizing the Task List Page

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

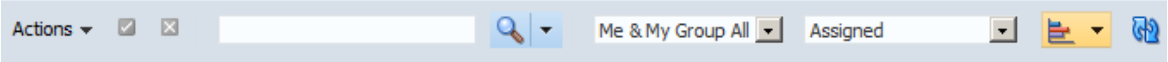
Note:

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.

How To Filter Tasks

Figure 32-4 shows the filter fields.

Figure 32-4 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 **Assignee** drop-down list, select from the following:

  – **Me**—Retrieves tasks directly assigned to the logged-in user

  – **My Group**—Retrieves the following:

    ◆ Tasks that are assigned to groups to which the logged-in user belongs

    ◆ 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

  – **Me & 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

  – **Me (Previously)**—Retrieves tasks that the logged-in user has previously updated or closed

  – **Me (Review Only)**—Retrieves task for which the logged-in user is a reviewer

  From the **My Staff Tasks** tab, select **Reportees**.

- **State**—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, then, 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 State

To filter tasks based on assignee or state:

Select options from the **Assignee** and **State** lists. The task list is automatically updated based on the filter selections.

To Filter Tasks Based on Keyword Search

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

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 How To Map Attributes, for more information.

1. Click **Advanced**.
2. (Optional) Check **Save Search As View**, provide a view name, and use the **Display** tab to provide other information, as shown in Figure 32-5 and Figure 32-6.

Figure 32-5 Worklist Advanced Search—Definition Tab

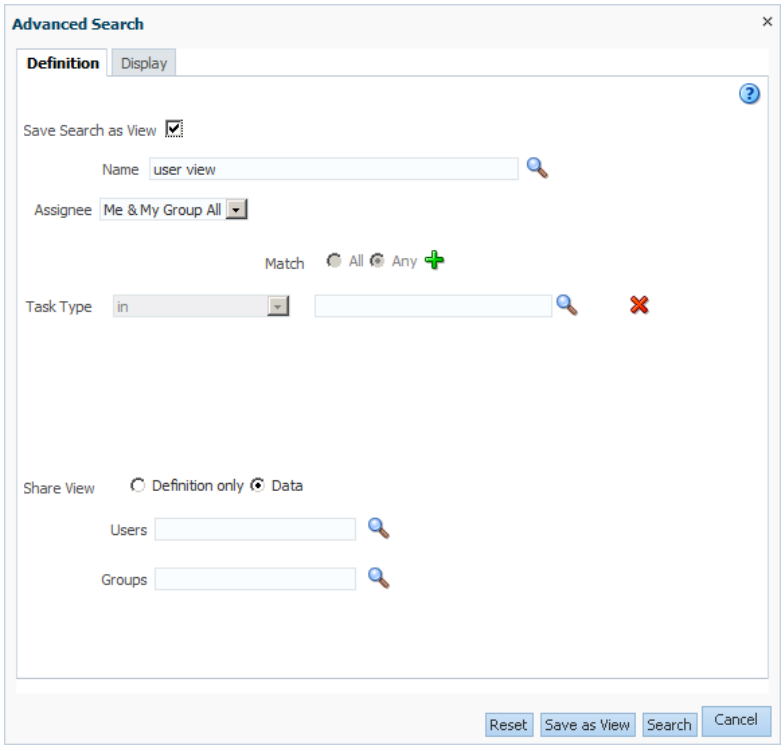


Figure 32-6 Worklist Advanced Search—Display Tab

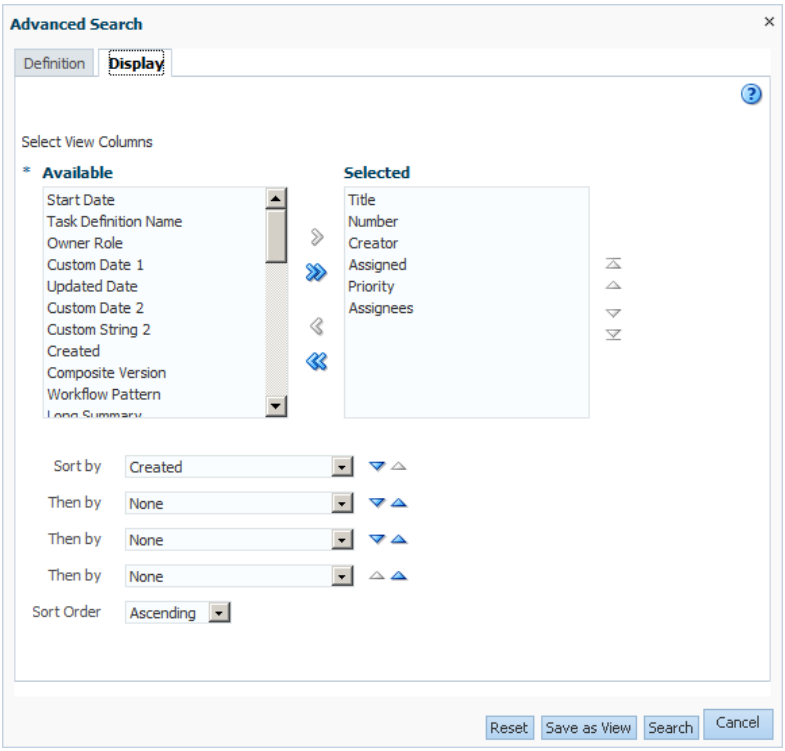


Table 32-4 describes the advanced search view columns available in the **Display** tab.

Table 32-4 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).

Table 32-4 (Cont.) Advanced Search—View Columns

Column	Description
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.
Partition	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.

Table 32-4 (Cont.) Advanced Search—View Columns

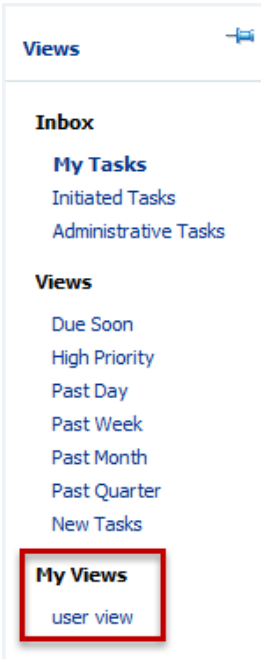
Column	Description
Assignees	The current task assignees (users, groups or application roles).
State	The state of the task instance.
Created	The date that the task instance was created.
Expires	The date on which the task instance expires.
Custom Date 1	Custom flex field 1 with Date data type
Custom Date 2	Custom flex field 2 with Date data type
Custom String 1	Custom flex field 1 with String data type
Custom String 2	Custom flex field 2 with String data type
Custom Number 1	Custom flex field 1 with Number data type
Custom Number 2	Custom flex field 2 with Number data type

The saved view appears in the Views pane under **My Views**, as shown in Figure 32-7.

Note:

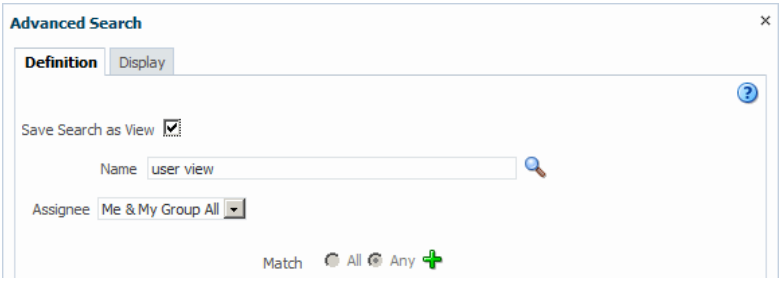
When a user view is created, and there are multiple versions of the same composite deployed, then selecting the task type with a particular version, for example, 'TestCompositeHumanTask2.0 ' does not ensure that only the tasks corresponding to this version are filtered. Instead use the task definition id column in the conditions, apart from selecting the task type, to get the correct result.

Figure 32-7 Saving a View



3. Select an assignee, as shown in Figure 32-8.

Figure 32-8 Worklist Advanced Search



4. Add conditions (filters), as shown in Figure 32-9.

Figure 32-9 Adding Filters for an Advanced Search on Tasks

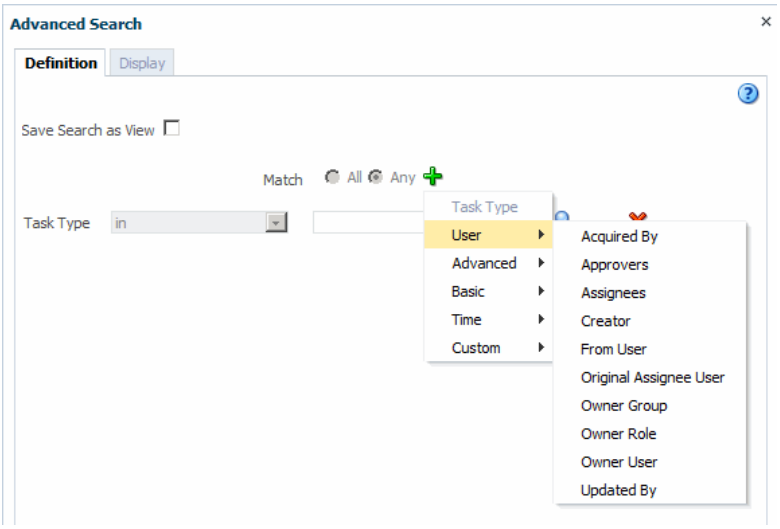


Table 32-5 describes the available conditions.

Table 32-5 Advanced Search—Conditions

Condition	Description
User Conditions	-
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.
Approvers	The approvers of the task.
Creator	The name of the creator of the task.
From User	The from user for the task.
Original Assignee User	The name of the user who delegated the task in the case when the user delegates a task to another user.
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.
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.
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.
Updated By	The user who last updated the task.
Advanced Conditions	-
Application Context	The application to which any application roles associated with the tasks (such as assignees, owners, and so on) belong.

Table 32-5 (Cont.) Advanced Search—Conditions

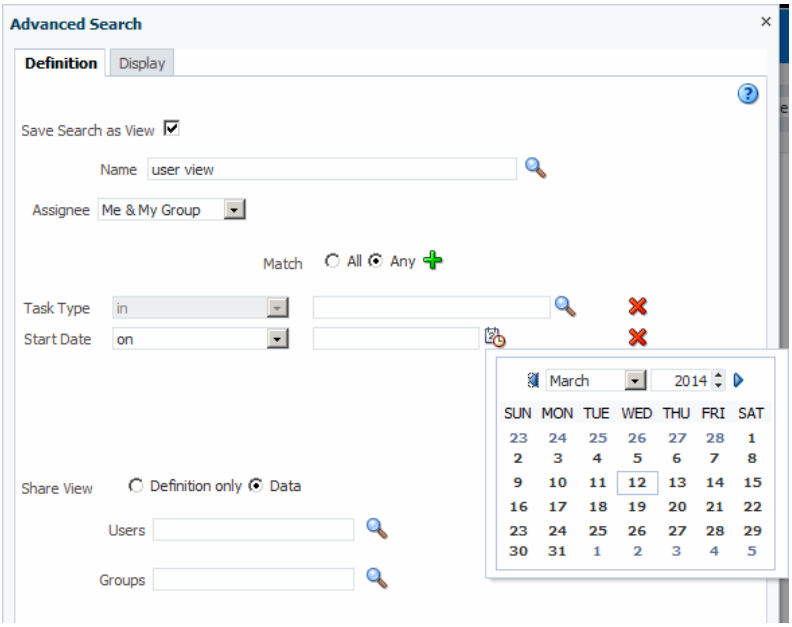
Condition	Description
Component	The name of the task component that defines the task instance.
Composite	The name of the composite that contains the task component that defines the task instance.
Composite Distinguished Name	The unique name for the particular deployment of the composite that contains the task component that defines the task instance.
Composite Version	The version of the composite that contains the task component that defines the task instance.
Partition	The domain to which the composite that contains the task component that defines the task instance belongs.
Task Display URL	The URL to display the details for the task.
Basic Conditions	-
Category	The category of the task.
Identifier	The (optional) custom unique identifier for the task. This is an additional unique identifier to the standard task number.
Number	An integer that uniquely identifies the task instance.
Outcome	The outcome of the task, for example Approved or Rejected. This is only set on completed task instances.
Percentage Complete	The percentage of the task completed (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.
State	The state of the task instance.
Task Definition Name	The name of the task component that defines the task instance.
Task Namespace	The namespace of the task.
Title	The title of the task.
Time Conditions	The category of the task.
Assigned	The date that this task was assigned.
Created	The date that the task instance was created.
Due Date	The due date of the task (used with ToDo tasks).
End Date	The end date of the task (used with ToDo tasks).
Expires	The date on which the task instance expires.
Start Date	The start date of the task (used with ToDo tasks).

Table 32-5 (Cont.) Advanced Search—Conditions

Condition	Description
Updated Date	The date that the task instance was last updated.
Custom Conditions	-
Custom Date 1	Custom flex field 1 with Date datatype
Custom Date 2	Custom flex field 2 with Date datatype
Custom String 1	Custom flex field 1 with String datatype
Custom String 2	Custom flex field 2 with String datatype
Custom Number 1	Custom flex field 1 with Number datatype
Custom Number 2	Custom flex field 2 with Number datatype

5. Select **Any** or **All** for matching multiple filters.
6. Add parameter values, shown in Figure 32-10.

Figure 32-10 Advanced Search



7. Specify whether to share either this view's definition or its data, and the users or groups to share it with.
8. Click **Search**.

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

How To Create, Delete, and Customize Worklist Views

The **Views** menu, shown in Figure 32-11, displays the following:

- **Inbox**—Shows all tasks that result from any filters you may have used. The default shows all tasks.

- **Standard Views**—Shows standard views and views that you defined.
- **My Views**—Shows views that you have created.

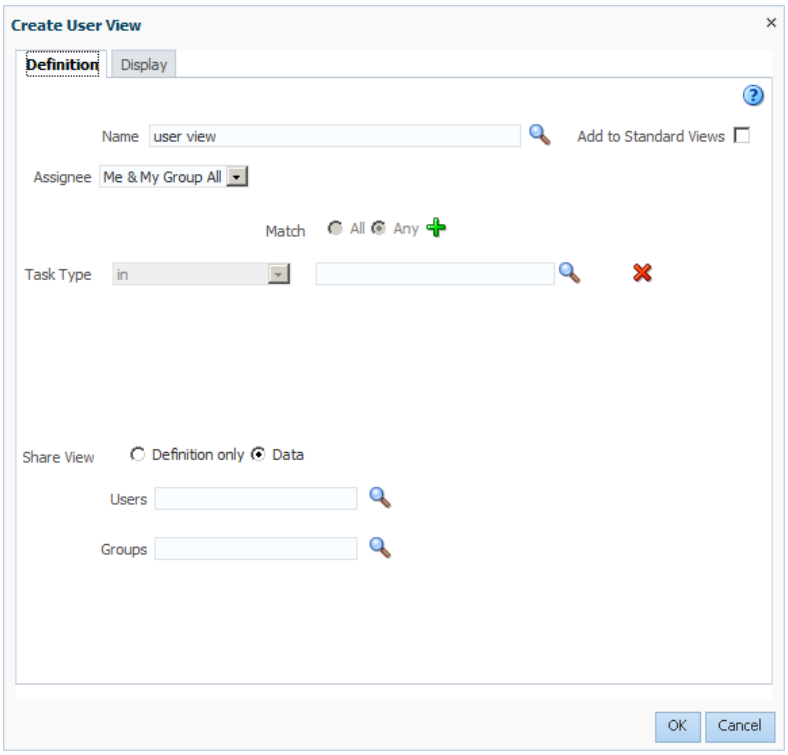
Figure 32-11 Worklist Views

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

To create a worklist view:

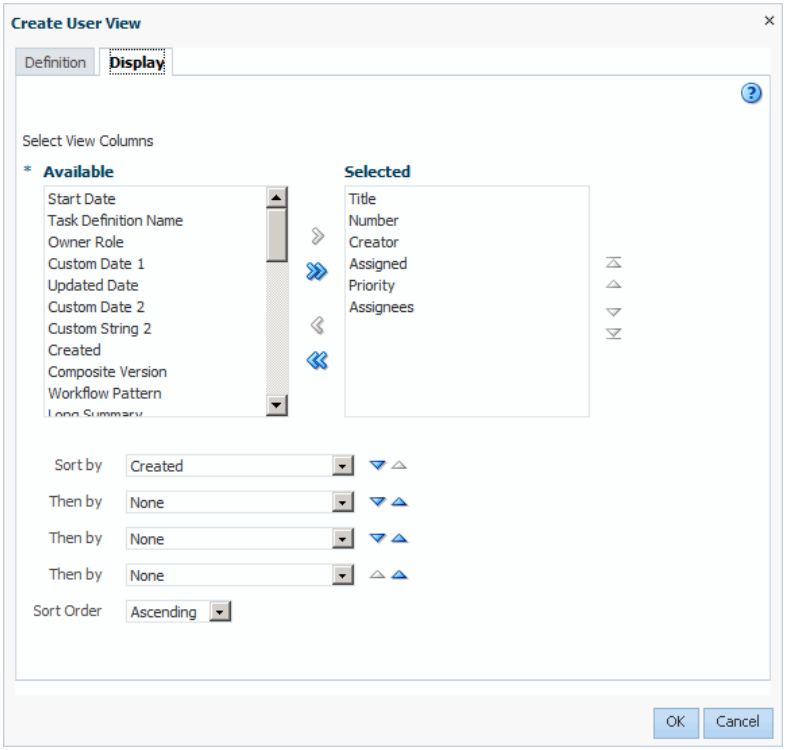
1. In the **Views** section, click **Add View**. The Create User View dialog box appears as shown in figure
2. Use the **Definition** tab of the Create User View dialog box, shown in Figure 32-12, to do the following:
   - **Name**—Specify a name for your view.
   - **Add to Standard Views**—This option 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 **Me**, **My Group**, **Me & My Group**, **Me (Previously), Me (Review Only)**, **Creator**, **Reportees**, **Admin**, **Owner**.
   - **Match**—Select **All** or **Any** to match the conditions you added.
   - **Add Condition (a plus sign)**—Select the conditions that apply to your view.
   - **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 **My Views**.
   - **Users**—Specify the users (grantees) who can share your view.
   - **Groups**—Specify the groups who can share your view.

Figure 32-12 Creating a Worklist View



3. Use the **Display** tab of the Create User View dialog, shown in Figure 32-13, to customize the fields that appear in the view.

Figure 32-13 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**—Select a column to sort on.
- **Then by**—Select a second column to sort on.
- **Then by**—Select a third column to sort on.
- **Then by**—Select a forth column to sort on.
- **Sort Order**—Select ascending or descending order.

4. Click **OK**.

   The saved view appears in the **Views** panel under **My Views**

To delete a view:

> **Note:**
>
> If an administrator inadvertently deletes the pre-seeded standard views, then those views do not remain permanently deleted. They are recreated when the server restarts.

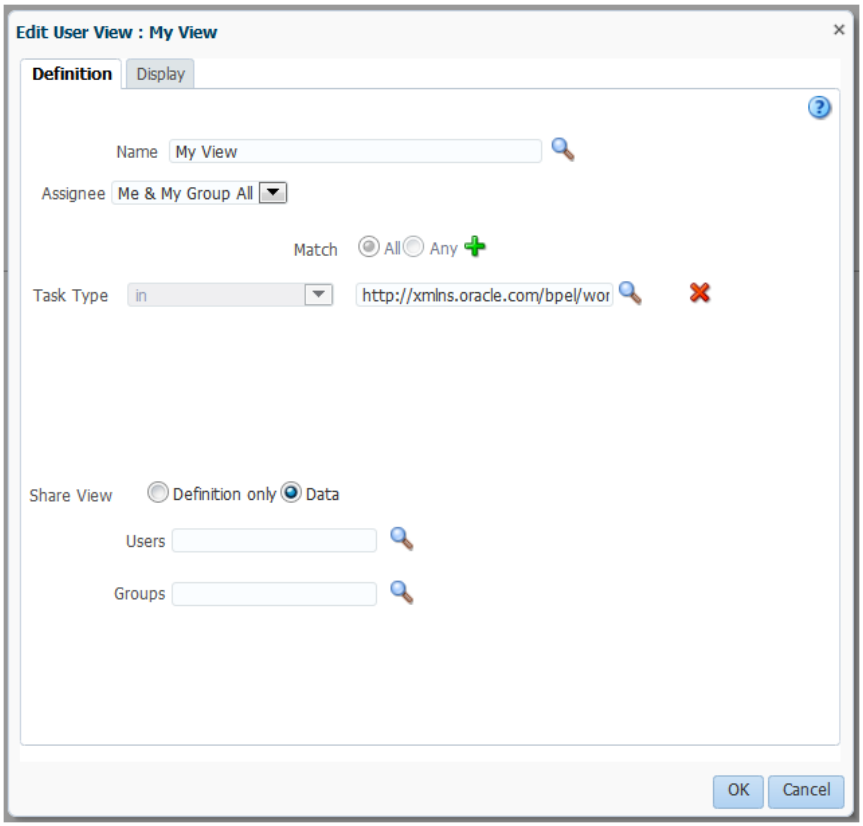
1. In the Views panel, select a view.
2. Click the **Delete** icon.
3. The Confirm Delete dialog box prompts you to confirm that you want to delete the view.
4. Click **Yes**. The view is deleted.

To Customize a Worklist View

To customize a worklist view:

1. In the **Views** pane, select the view you want to customize.
2. Select **Edit View**. The Edit User View dialog box appears.
3. Use the items in the Edit User View dialog box to customize the view, as shown in Figure 32-14, and click **OK**.

Figure 32-14 Customizing Fields in a Worklist View



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:

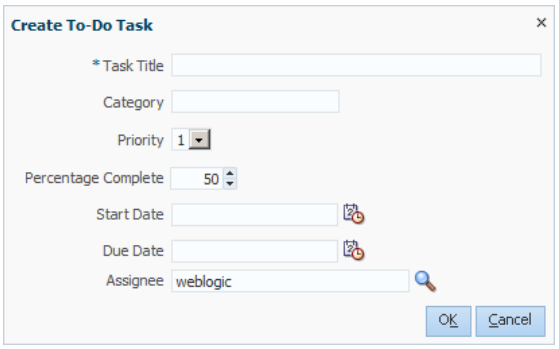
1. Click the **Edit** icon.
2. Add or remove status states for display, as shown in Figure 32-15, and click **OK**.

Figure 32-15 Customizing the Task Status Chart

How To Create a ToDo Task

Use the Create ToDo Task dialog, shown in Figure 32-16, to create a top-level ToDo task for yourself or others. This task is not associated with a business task.

Figure 32-16 The Create ToDo Task Dialog



To-Do 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.

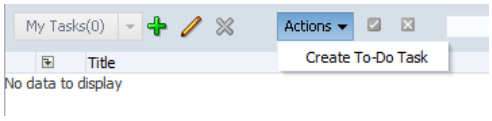
ToDo tasks can be reassigned, escalated, and so on, and deleted (logical delete) and purged (physical delete). Reassignment, escalation, and so on of the parent task does not affect the assignment of any child ToDo tasks. The completion percentage of a ToDo task can be reset to less than 100% after it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

1. From the **Actions** list, select **Create To-Do Task**, as shown in Figure 32-17.

   Figure 32-17 Creating a To-Do Task

2. Provide details in the Create ToDo Task dialog, shown in Figure 32-16, and click **OK**.

   - **Task Title**: Enter anything that is meaningful to you.
   - **Category**: Enter anything that is meaningful to you.
   - **Priority**: Select from **1** (highest) to **5** (lowest)
   - **Percentage Complete**: This attribute indicates how much of the task is completed. 100% sets the attribute as completed.

- **StartDate**: The task start date. The start date need not be the current date.
- **Due Date**: The due date does not trigger an expiration. You can also see overdue tasks.
- **Assignee**: You can assign yourself or someone else.

How to Create Subtasks in Worklist Application

A subtask is a child of a parent task, either a ToDo task or a business task. Creating a subtask can be helpful, for example, when a purchase order contains several line items and you need a separate approval process for one of them.

What You May Need to Know About Creating Subtasks

Here are some things to keep in mind when creating subtasks:

- A ToDo task can have only one level of subtasks. When all ToDo subtasks are 100% complete, the parent ToDo task is also marked as complete. If the parent ToDo task is completed, then ToDo subtasks are at 100% within the workflow system.

  If the parent is a business task and that task is completed, then the subtasks of that task are withdrawn.

- If you explicitly set a ToDo task to 100%, there is no aggregation on the parent task.
- If you are using a release of Oracle Business Process Management that is before 11*g* Release 1 (11.1.1.7.0), then you must re-create the task form for any task for which you are creating a subtask. You may, however, continue to use processes that were deployed in earlier releases.

  If you do not re-create the task form, then the **Actions** list in the task form itself does not provide the option to create a subtask. You can, however, create a subtask by selecting **Create Subtask** from the **Actions** list above the worklist.

- If you are the administrator for the BPMN service engine, be aware that subtasks do not appear in the Oracle Enterprise Manager Fusion Middleware Control.

To create a subtask:

1. In the worklist, select the task for which you want to create a subtask.
2. From the **Actions** list, select **Create Subtask**.

   The Create Subtask dialog box appears.

3. In the Create Subtask dialog, define the subtask, keeping the following in mind:.

   - **Title** is a required field.
   - If there is more than one available form for this subtask, then the **Form** field provides a list for your selection. Otherwise, the **Form** field shows the name of the default form. You can use a task form different from the one associated with the parent task.
   - Possible routing types are:

     – Single Approver

  - Group Vote, also referred to as a parallel task. For this routing type, you are prompted to enter multiple participants.
  - Chain of Single Approvers, also referred to as a sequential task. For this routing type, you are prompted to enter multiple participants.

- You specify participants by performing a search and selecting from the results. You can select multiple users, groups, or application roles.

4. When you have finished specifying the subtask, in the Create Subtask dialog box, click **OK**. This refreshes the task list. When you select the parent task, the Task Details page now includes a Subtasks section displaying the details about the subtask you created for that task.

Note:

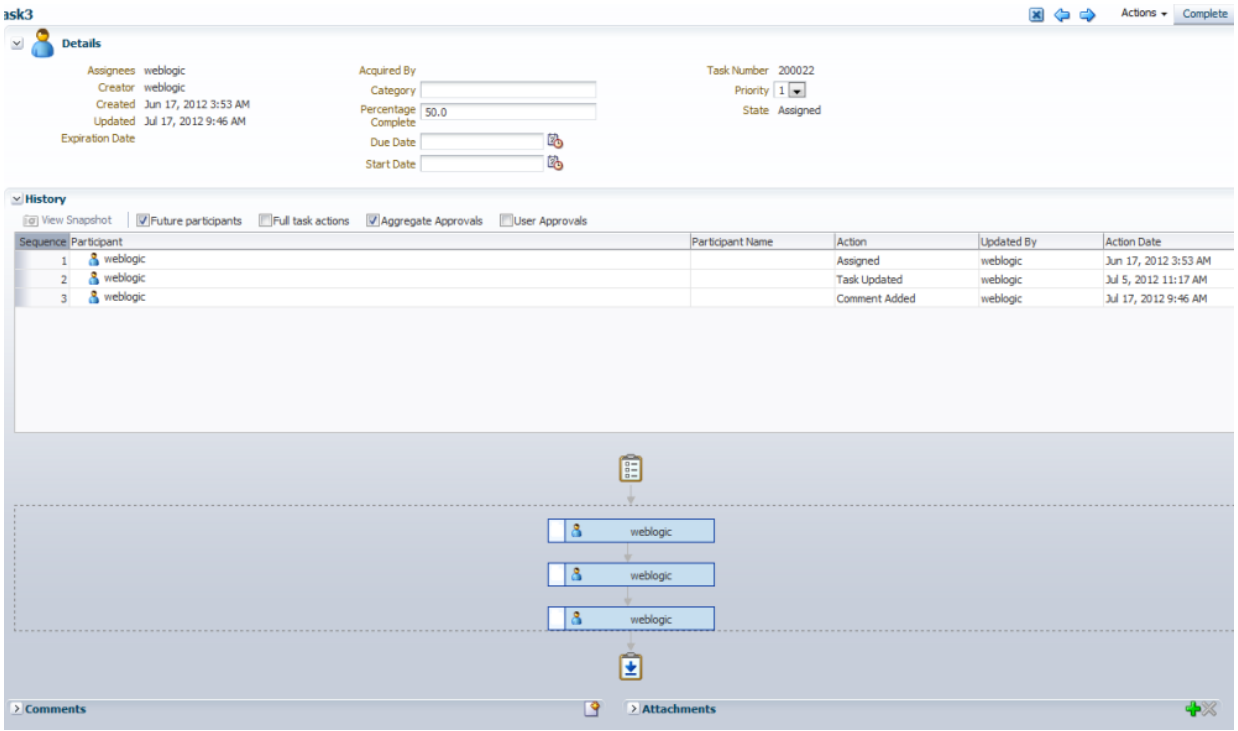
- If you specified more than one participant for the subtask, then the Subtask region displays a separate item for each participant.
- If a participant completes a subtask, then you must manually refresh the task to show the details for that completed subtask.

Acting on Tasks: The Task Details Page

Task details can be viewed inline (see the lower section in Figure 32-2) or in the same window or a new window. (Modify settings in Edit Inbox Settings.)

Figure 32-18 shows the task details page.

Figure 32-18 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:

- 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.
- History—Displays the approval sequence and the update history for the task. See Task History, for more information.

  Table 32-6 tells what the icons used in the Task Details History section signify.

Table 32-6 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.

Note:

In an environment with servers clustered for high availability purposes, file uploading is not supported if a failover occurs. If the active server shuts down, then the uploading process is not assumed by the other server and the upload fails.

Comments and attachments are shared between tasks and subtasks. For example, when you create a ToDo task and add comments and attachments, subtasks of this ToDo task include the same comments and attachments.

The Task Details page may appear differently depending on the tool used during design time to develop the task form it displays.

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.

Note:

Certain functions, such as restricted task reassignment, are available only when a single task is selected. If multiple tasks that use restricted reassignment are selected, then the restricted reassignment algorithm is not invoked. In that case, the complete list of users gets returned as though restricted reassignment had not been specified.

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.

System Actions

The action bar displays system actions, which are available on all tasks based on the user's privileges. Table 32-7 lists system actions.

Table 32-7 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. • Pushback is designed to work with single approvers and not with group votes. Pushback from a stage with group vote (or parallel) scenario to another stage is not allowed. Similarly, you cannot push back from a single assignee to a group vote (or parallel) scenario.
Reassign	If you are a manager, you can delegate a task to reportees.
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).

Table 32-7 (Cont.) System Task Actions

Action	Description
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.
Start/Stop Task	When the user chooses to start or stop work on the task the time stamp is assigned to all the tasks selected. It is used to calculate the working durations of the task. The user can use Start/Stop Task multiple times on the same task, for example startTask -> stopTask -> startTask -> stopTask -> startTask -> completeTask The total working duration is the sum of all of these time intervals. Start/Stop Task operations are only available for tasks in Assigned or Request Information status. Start/Stop Task is not available for Aggregated Task.

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

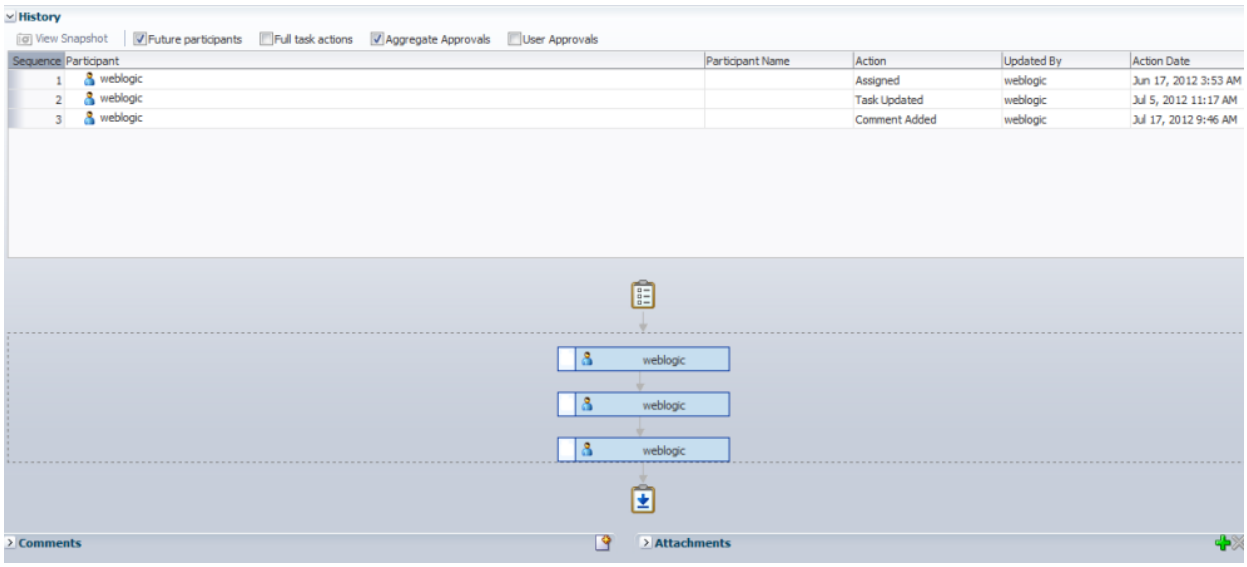
You can include the following actions in the short history list by modifying the `shortHistoryActions` element.

- 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

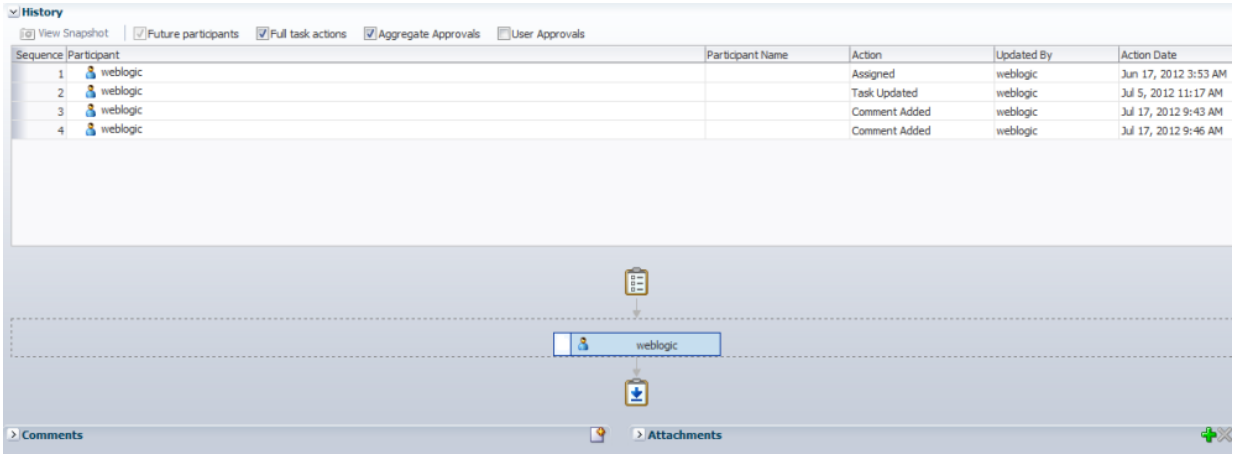
The history provides a graphical view of a task flow, as shown in Figure 32-19.

Figure 32-19 History: Graphical View



Check **Full task actions** to see all actions performed, including those that do not make changes to the task, such as adding comments, as shown in Figure 32-20.

Figure 32-20 History: Full Task Actions



Available ways to view the task history include:

- Take a task snapshot
- See future approvers
- See complete task actions
- Aggregate tasks

Note:

The history of a parent task also displays the history of any subtasks it contains.

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 navigate 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.

The task list does not display the actions for a task. A user has to take action from the task details.

If a human task was set up to require a password, then when you act on it, you must provide the password.

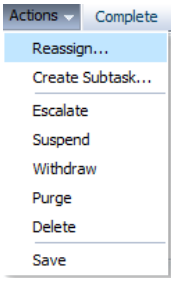
Note:

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 **Actions** list, select **Reassign**, as shown in Figure 32-21.

Figure 32-21 Reassigning a Task

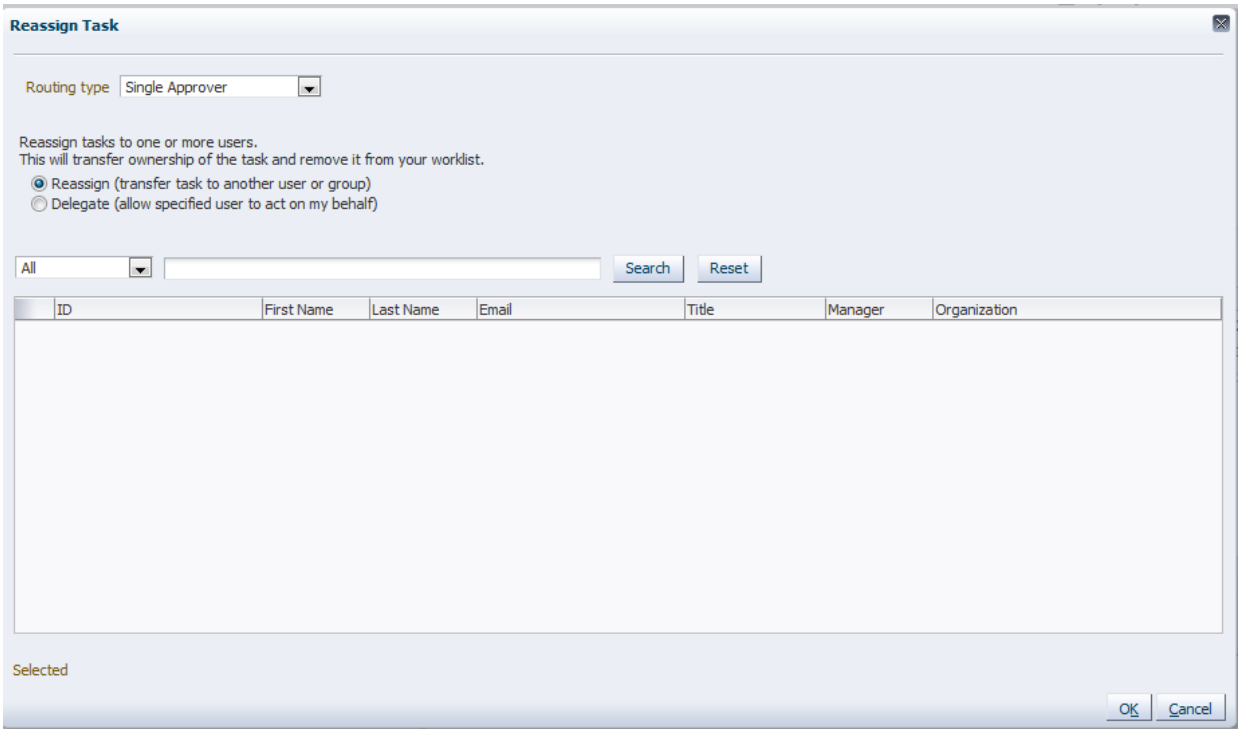


2. Select **Reassign** or **Delegate**.

   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 32-22.

Figure 32-22 Reassigning a Task

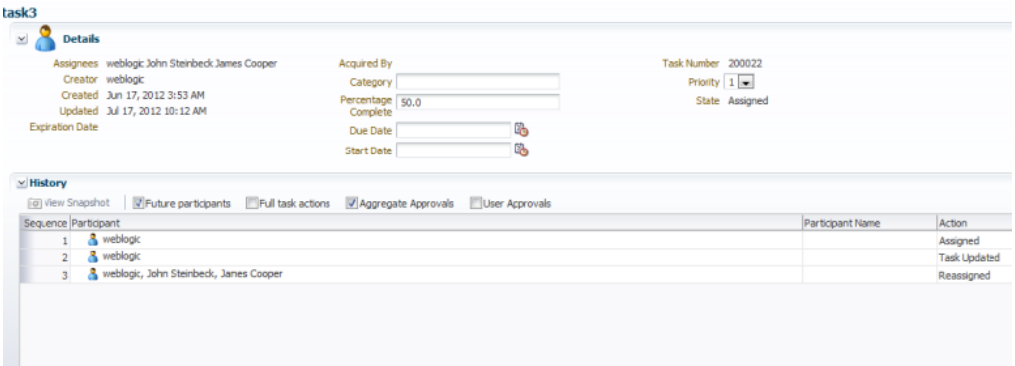


A supervisor can always reassign tasks to any of his reportees.

4. Select the names by clicking the check box and click **OK**.

   You can reassign to multiple users or groups. One of the assignees must claim the task, as shown in Figure 32-23.

Figure 32-23 Claiming a Task



Note:

When task details have been upgraded from an earlier release, you can see a "Request Failed" error when executing the **Reassign** action. Actually, the reassign completes, and when you click OK again, a popup says the task is already assigned.

To eliminate the error message, upgrade your task flow applications by opening them in Oracle JDeveloper, then redeploy the task form.

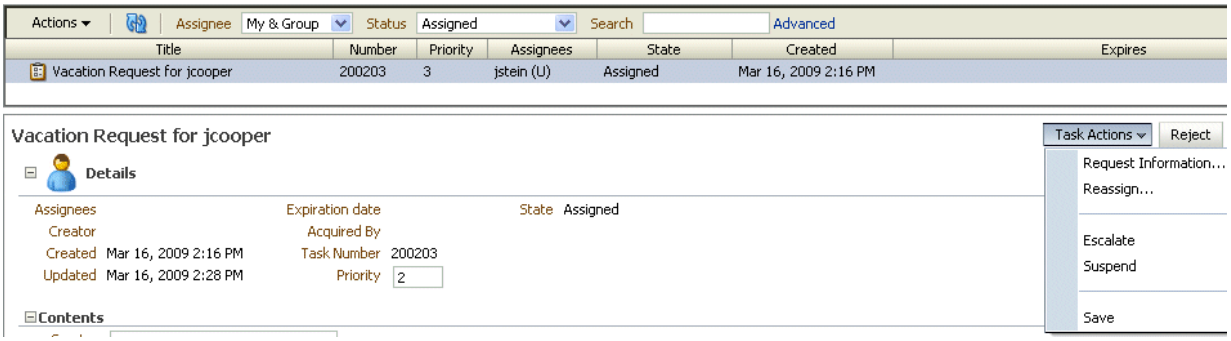
To Request Information

To request information:

1. From the **Actions** list, select **Request Information**, as shown in Figure 32-24.

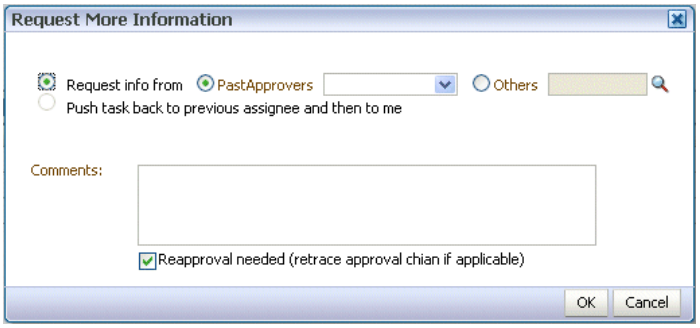
   This action is available only when you enable the "Allow participants to invite other participants" setting in the task definition.

Figure 32-24 Requesting Information



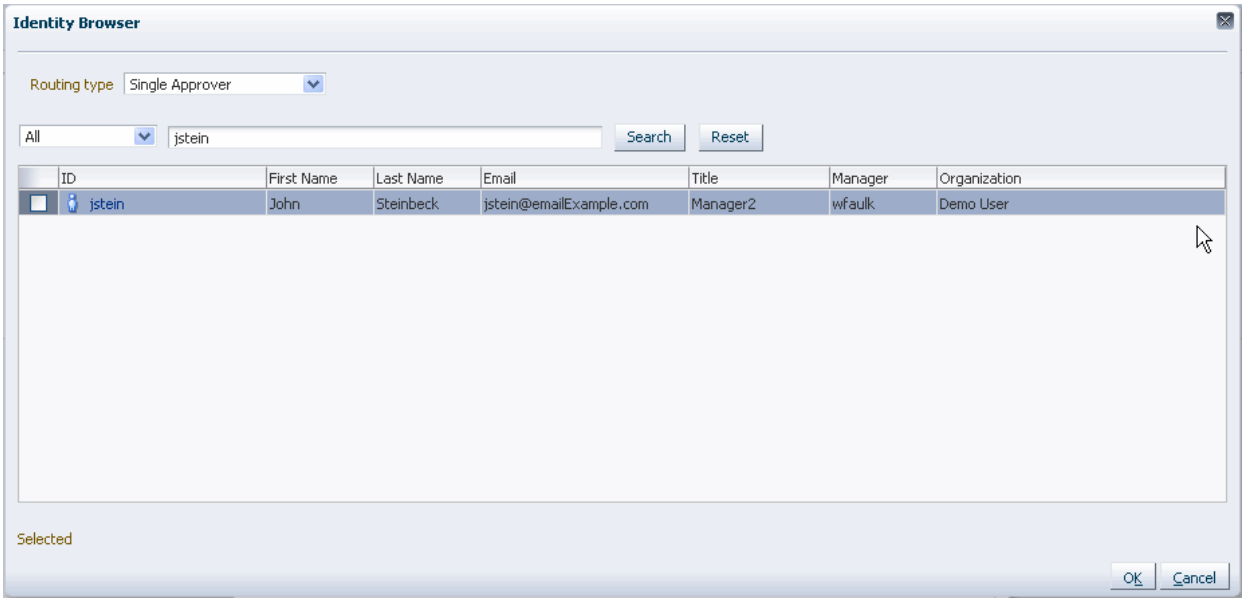
2. Request information from a past approver or search for a user name, or push the task back to the previous assignee, as shown in Figure 32-25.

   Figure 32-25 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 32-26.

Figure 32-26 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 search by using jstein.

---

3. Click **OK**.

To Route a Task

Note:

The task definition must be set to **Allow participants to invite other participants** before the task can be routed.

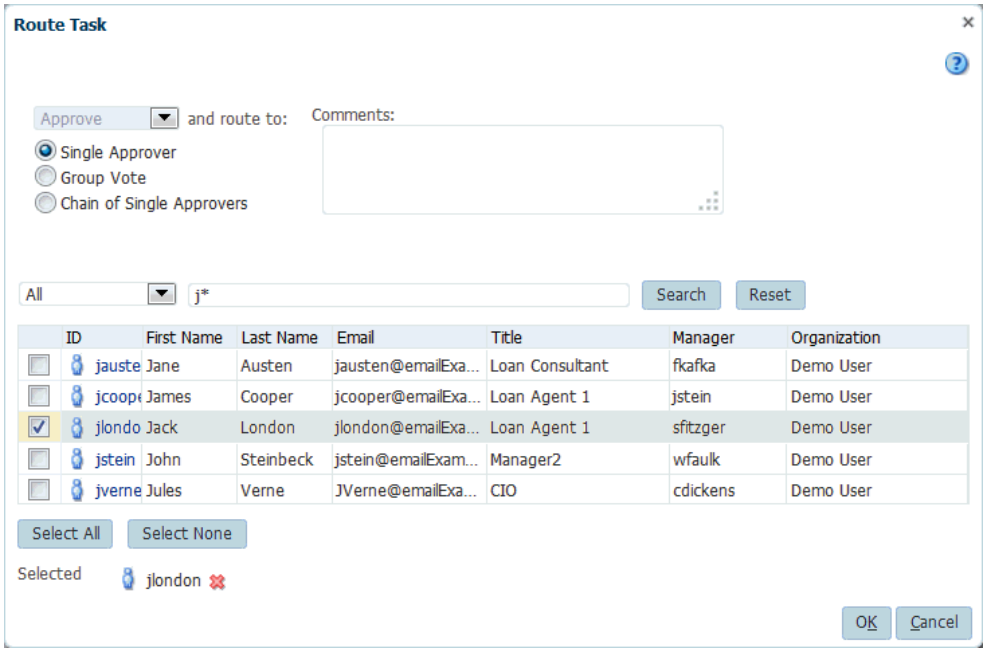
To route a task:

1. From the **Task Actions** list, select **Adhoc Route**, as shown in Figure 32-27.

Figure 32-27 Ad Hoc Routing

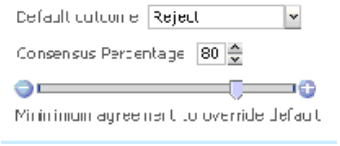
2. Select an action and a routing option, as shown in Figure 32-28.

Figure 32-28 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 32-29.

Figure 32-29 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

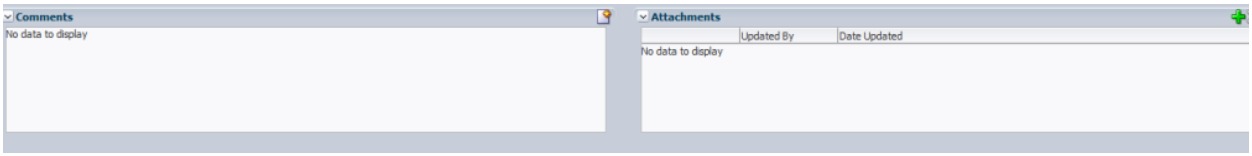
To add comments or attachments:

Note:

- If you are the initiator of the task, then your comment is shared with all process participants and not only with task assignees. The option to share with only task participants is not available to you.
- Comments added to a parent task also appear in any subtasks of that parent.
- 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**.
- In an environment with servers clustered for high availability purposes, file uploading is not supported if a failover occurs. If the active server shuts down, then the uploading process is not assumed by the other server and the upload fails.
- If you are using an ADF connection and you receive a "No Protocol" error when attempting to add an attachment, verify that your `connections.xml` file is synchronized with the correct WSDL file. The `connections.xml` file is located in the directory `.adf/META-INF/` in your ADF workspace.

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

Figure 32-30 Worklist Comments and Attachments

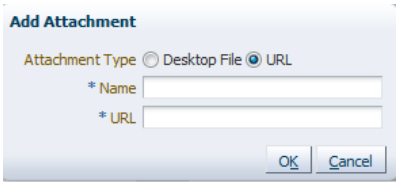


2. Enter comment text and click **OK**. Comments cannot be deleted after they are added.

   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 32-31, and click **OK**.

Figure 32-31 Adding a Worklist Attachment



   If you attach a URL file in Oracle BPM Worklist (for example, `http://www.example.com/technology/products/oem/management_partners/snmpwp6.gif`), it is not sent as an email attachment. Instead, it appears as a link in the task details of the email 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**.

How To Act on Tasks That Require a Digital Signature

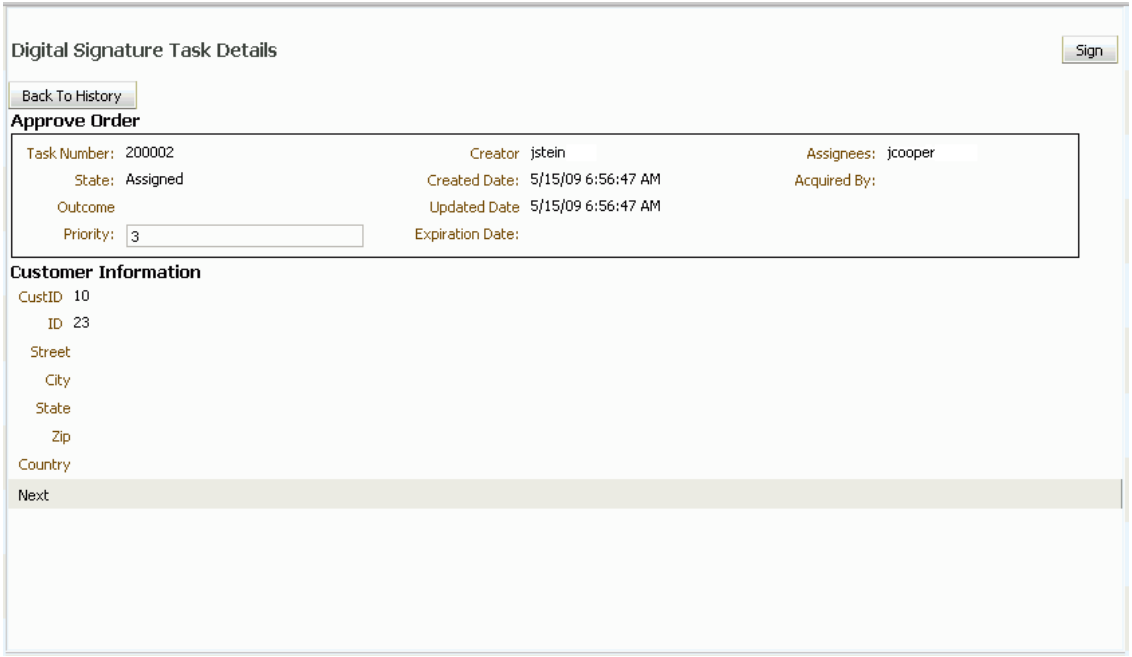
The worklist supports the signature policy created in the human task:

- **No signature required** — Participants can send and act on tasks without providing a signature.
- **Password required** — Participants mus tspecify 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.

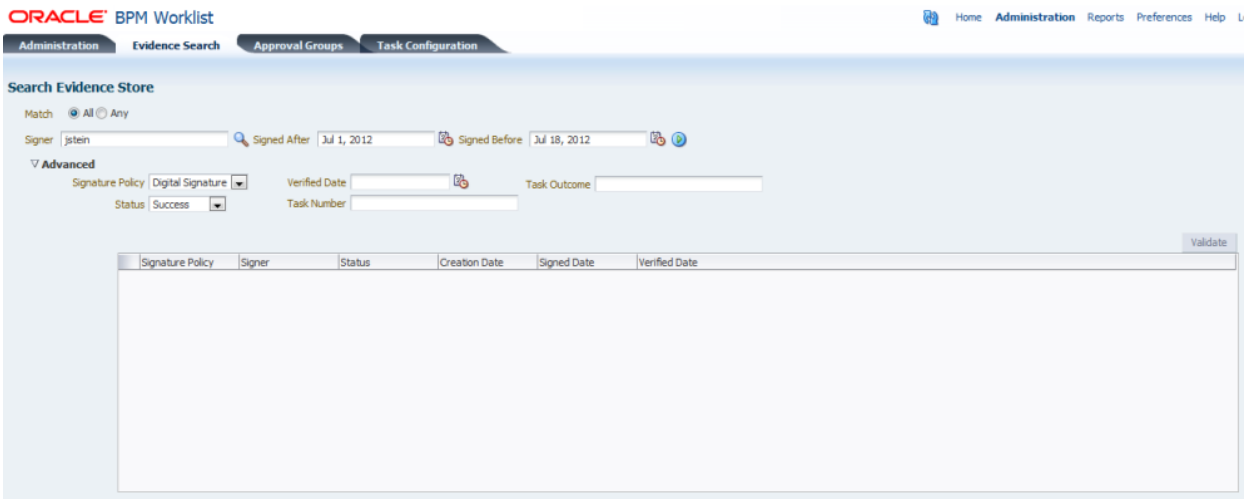
When you act on a task that has a signature policy, the **Sign** button appears, as shown in Figure 32-32.

Figure 32-32 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 32-33.

Figure 32-33 The Evidence Store



See 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. Select the **Certificates** link.
3. Upload the certificate you want to use to sign your decision, as shown in Figure 32-34.

   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:

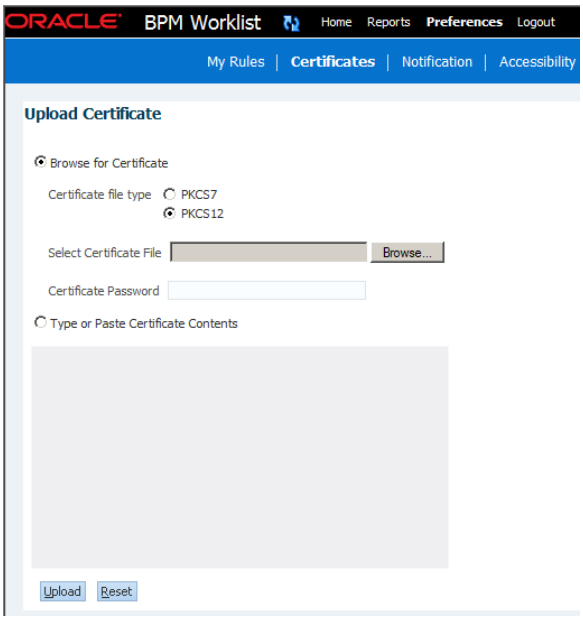
   a. Export the chain from Mozilla Firefox as a .P12 file (PKCS12 format that also contains your private key).

   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 32-34 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 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 Evidence Store Service and Digital Signatures.

Approving Tasks

Table 32-8 describes the type of actions that can be performed on tasks by the various task approvers.

Table 32-8 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	Yes[1]	No	No	No
Delegate	No	No	Yes	No	No	No
Delete	No	No[2]	Yes[2]	Yes[2]	No	No
Error	No	No	Yes[3]	No	No	No
Escalate	Yes	Yes[4]	Yes	No	No	No
Info Request	No	No	Yes	No	No	No

Table 32-8 (Cont.) Task Actions and Approvers

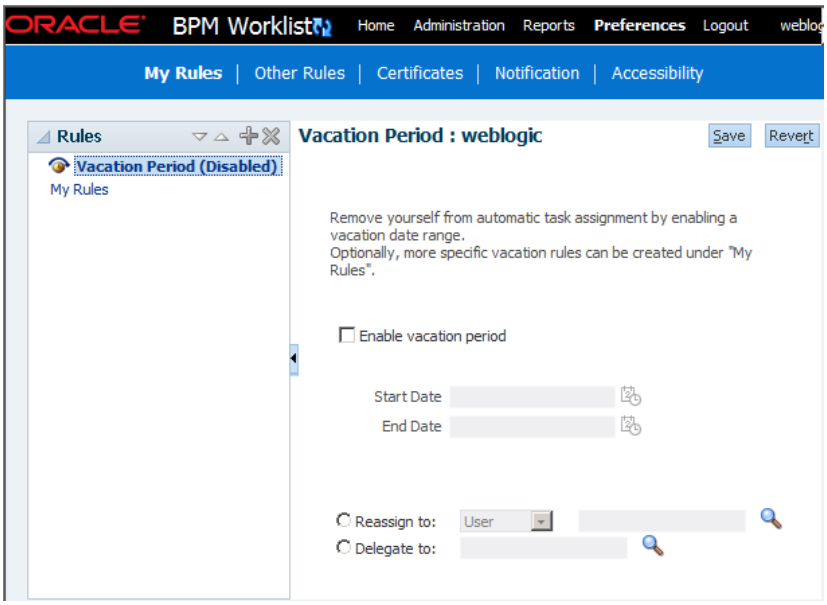
Task Action	Admin	Owner (+ Owner Group)	Assignee (+ Assignee Manager + Assignee Group + Proxy Assignee)	Creator	Reviewer	Approver
Info Submit	No	No	Yes	No	No	No
Override Routing Slip	Yes	Yes	No	No	No	No
Push Back	No	No	Yes	No	No	No
Purge	Yes[2]	Yes[2]	No	Yes	No	No
Reassign	Yes	Yes[5]	Yes (No for proxy 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

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 32-35.

Figure 32-35 Setting a Vacation Period



Vacation rules are not executed for ToDo tasks. See 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.

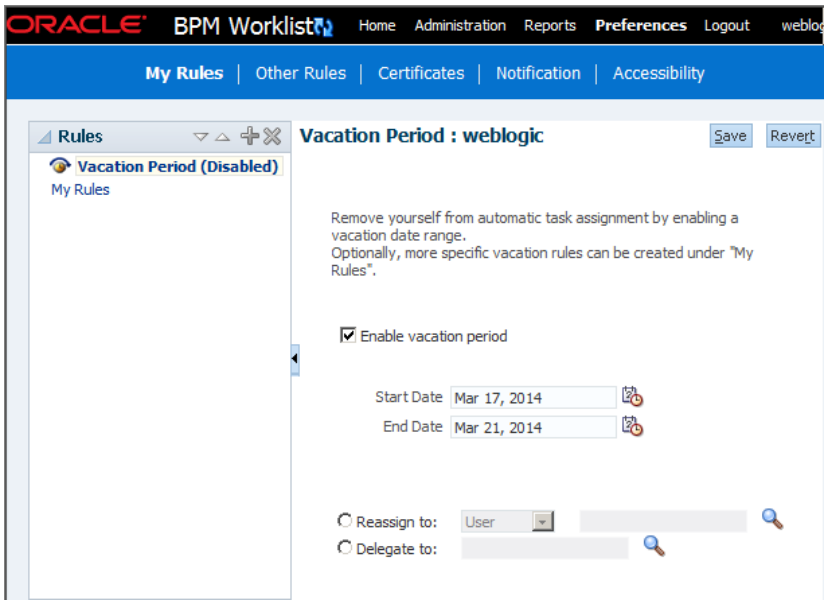
2. Select **Enable vacation period**.

3. Provide start and end dates.

4. Click **Save**.

   The vacation period is enabled, as shown in Figure 32-36.

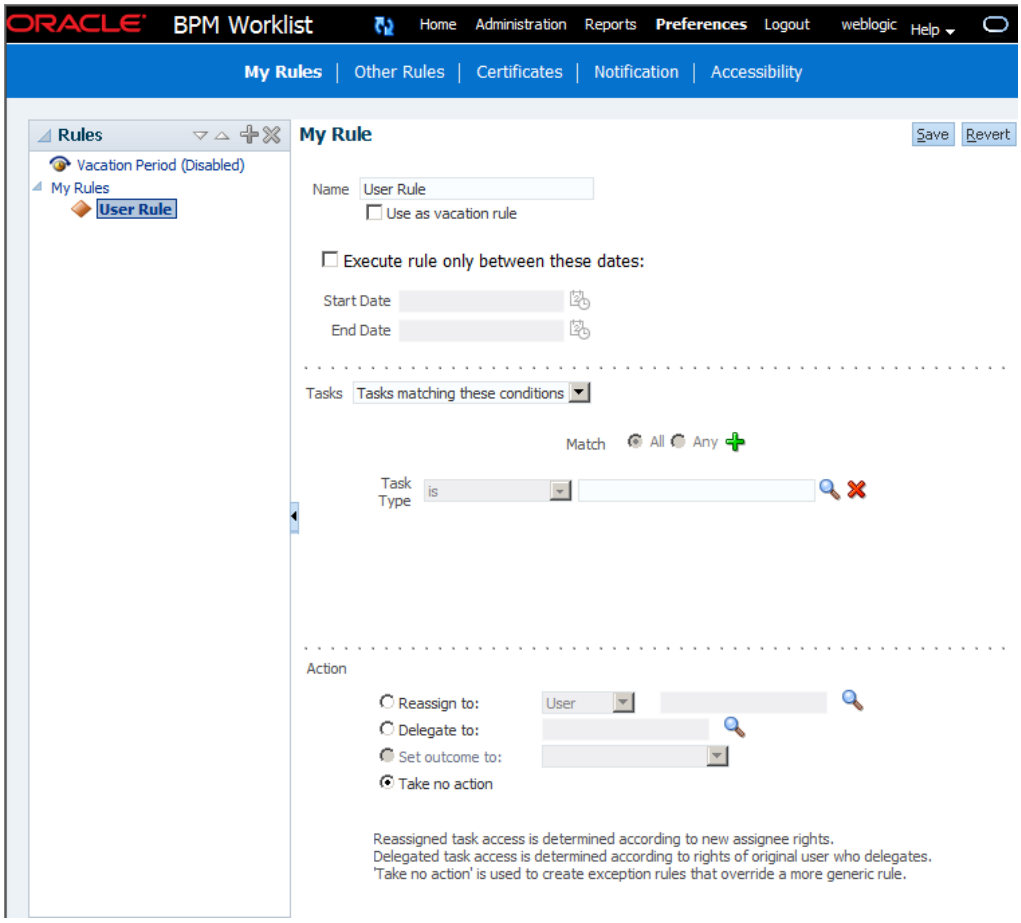
Figure 32-36 Enabling a Vacation Period



Setting Rules

Rules act on tasks, either a specific task type or all the tasks assigned to a user or group. Figure 32-37 shows where you set rules, including vacation rules (different from the vacation period settings described in Setting a Vacation Period).

Figure 32-37 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 32-37.

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

How To Create User Rules

Specify the following when creating a user rule:

- Rule name
- If the rule is a vacation rule. See 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 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** pane, 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.
   - Select **Execute rule only between these dates** and provide rule execution dates.
   - In the **TASKS** area, select All Tasks or Tasks matching these conditions. Then use the Add button to add rule conditions.
   - Browse for task types to which the rule applies.
   - In the **ACTION** area, select actions to be taken: **Reassign to**, **Delegate to**, **Set outcome to**, or **Take no action**), as shown in Figure 32-37.

   The new rule appears under the **My Rules** node.

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:

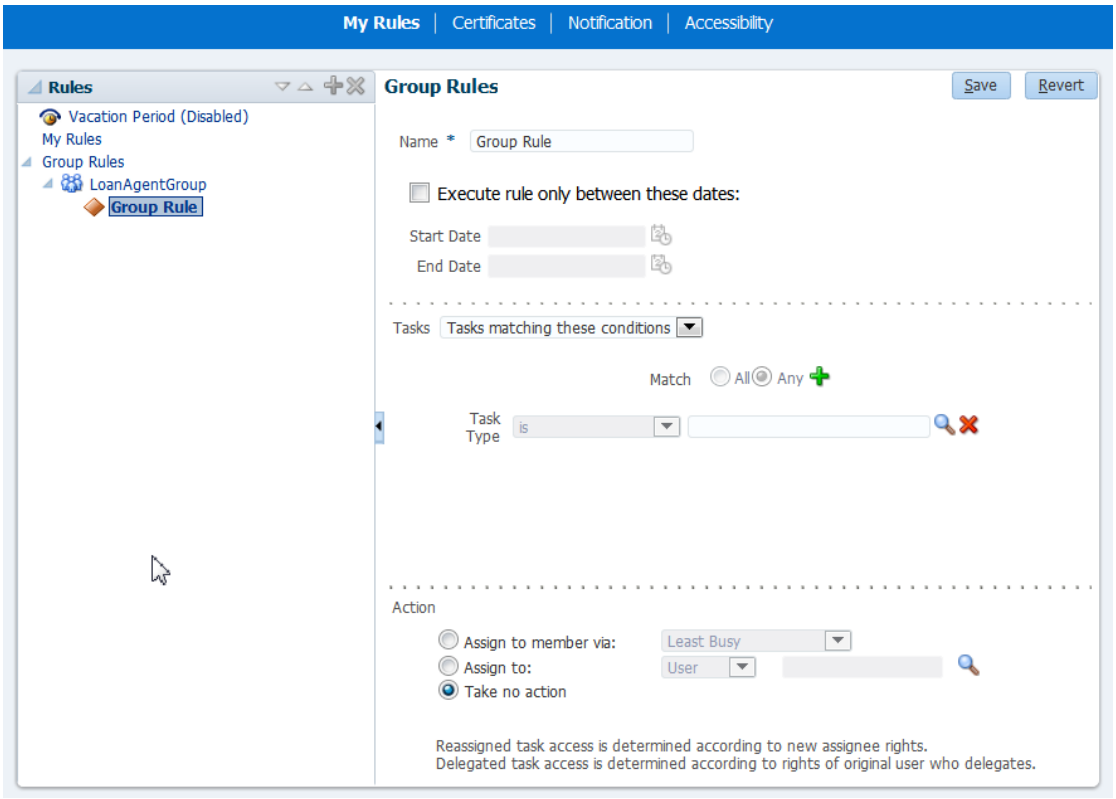
- **Assign to member via**—You can specify a criterion to determine which member of the group gets the assignment. This dynamic assignment criterion can include round-robin assignment, assignment to the least busy group member, or assignment to the most productive group member. You can also add your 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 New Rule**, as shown in Figure 32-38.

Figure 32-38 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 **TASKS** area, add rule conditions.
   - In the **ACTION** area, select the actions to be taken (or none) (**Assign to member via**, **Assign to**, or **Take no action**), as shown in Figure 32-38.

   The new rule appears under the **Group Rules** node.

How to Avoid Circular Logic in Reassigned Vacation Rules

When creating vacation rules, ensure that approval tasks are not reassigned in a circular fashion.

For example, jstein is jcooper's manager, and tasks that need to, go to jstein for approval. If jstein creates a vacation rule, ensure that those tasks will not be re-assigned to jcooper.

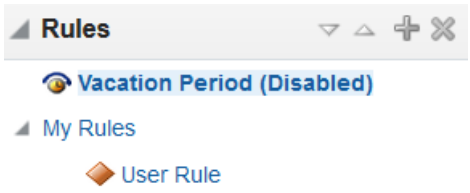
Alternatively, you can use the delegate option in vacation rules instead of reassign. For more information about the delegate option, see Reassigning and Delegating Tasks in Process Workspace.

How To Avoid Circular Dependency

In this example, we define two rules for User2, who wants to set a vacation period and reassign his tasks to someone else. The first rule states that if the task is not coming from User1, then reassign it to User1. The second rule states that if the task is coming from User1, then reassign it to User3.

The task flow is: jcooper to jstein to wfaulk. The rules are set for jstein.

1. Login to Worklist as `jstein` and go to the **Preferences** page.
2. Ensure that the Vacation Period is disabled, as shown below:



3. Click **My Rules** and click **Add New Rule +**.
4. Enter a name for the rule, for example `VacationRule1.`
5. Clear the **Use as vacation rule** check box.
6. Check the **Execute rule only between these dates** check box and enter the appropriate dates.
7. In the **Tasks** drop down, choose **Tasks matching these conditions.**
8. Click **Add condition +** and select `User, From User.`
9. In the new row, select `isn't.`
10. In the text box next to it, enter `jcooper` for the user name.
11. In the **Action** section, select **Reassign to** and enter `jcooper` for the user name.
12. Click **Save.**

    Repeat the steps above to create another rule with these inputs:

13. Enter a name for the second rule, for example VacationRule2.
14. Select the same start and end dates as in VacationRule1.
15. Add a **From User condition** of `is.`
16. Enter `jcooper` for the user name.
17. In the **Action** section, select **Reassign to** and enter `wfaulk` for the user name.
18. Click **Save.**

Invoke the composite and the tasks will be assigned as expected based on these two new rules defined for jstein.

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).

Using the Worklist Administration Functions

Administrators are users who have been granted the BPMWorkflowAdmin role. Administration functions include the following:

- Managing other users' or groups' rules
- Setting the worklist display (application preferences). Application preferences customize the appearance of the worklist, including:
  - The login realm label
  - The resource bundle
  - Where the language locale information is retrieved from
  - The branding logo
  - The branding title
  - The branding skin
  - Any external applications you want to use, for example, `oracle.com`, or `google.com`
- Specifying mapped attributes

An administrator can view and update all tasks assigned to all users. An administrator's **Assignee** filter displays **Admin** when the Admin tab is selected.

This section contains these topics:

- How To Manage Other Users' or Groups' Rules (as an Administrator)
- How to Specify the Login Page Realm Label
- How to Specify the Resource Bundle
- How to Specify the Language Locale Information

- How to Specify a Branding Logo
- How to Specify the Branding Title
- How to Choose a Skin
- How to Enable Customized Applications and Links

For information about specifying mapped attributes, see Using Mapped Attributes (Flex Fields)

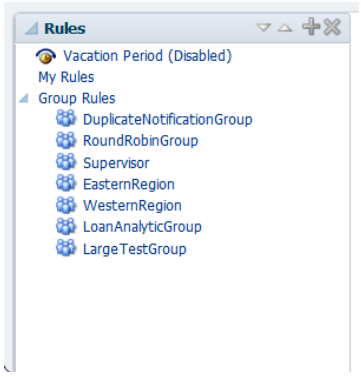
How To Manage Other Users' or Groups' Rules (as an Administrator)

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. Click the **Preference**s 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 32-39.

Figure 32-39 Creating Rules for Another User or Group



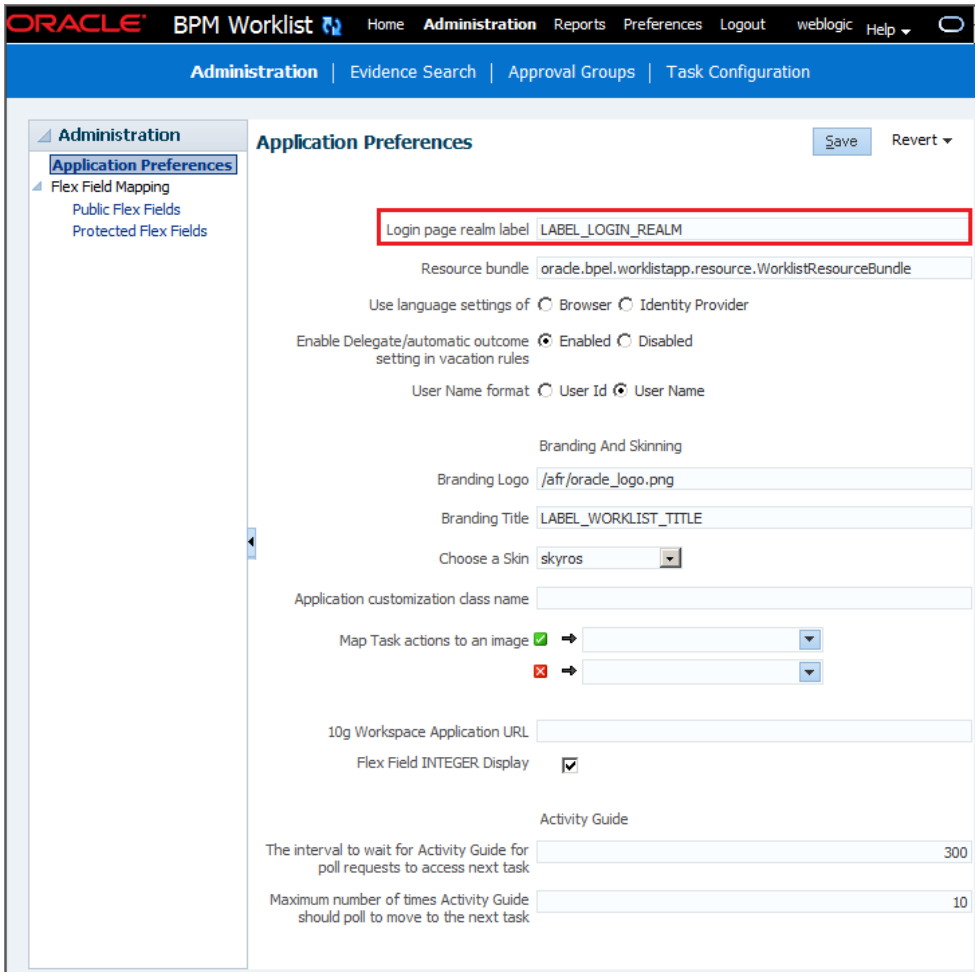
4. Click a user rules node, or click a group name (for a group rule).
5. Click the **Add** icon to create a rule.
6. Provide rule information, as shown in Figure 32-38, and click **Save**.

How to Specify the Login Page Realm Label

If the identity service is configured with multiple realms, then, when a user logs in to Oracle BPM Worklist, the login page displays a list of realm names. LABEL_LOGIN_REALM specifies the resource bundle key used to look up the label to display these realms. You can change the term *realm* to fit the user community—terms such as *country*, *company*, *division*, or *department* may be more appropriate. To change the term *realm*, customize the resource bundle, specify a resource bundle key for this string, and then set the **Login page realm label** parameter to point to that resource bundle key.

Figure 32-40 shows the Application Preferences page with the **Login page realm label** field highlighted. You reach the Application Preferences page by clicking **Administration** on the global toolbar at the very top of the Worklist Application interface.

Figure 32-40 Specifying the Login Page Realm Label



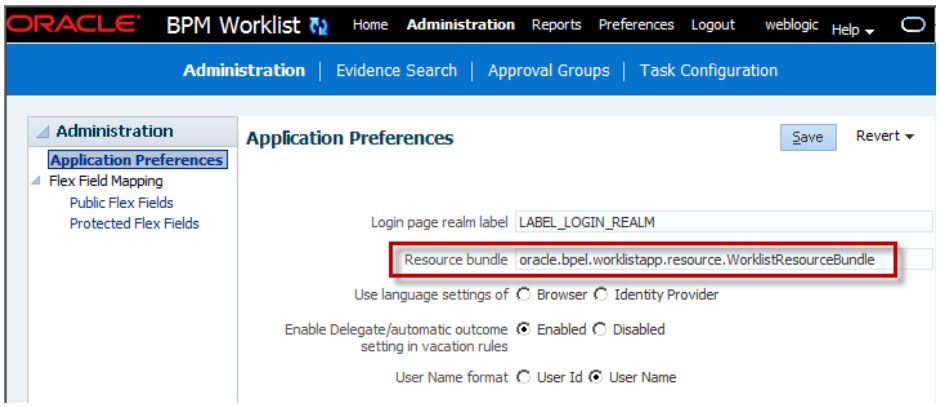
How to Specify the Resource Bundle

The resource bundle provides the strings displayed in the Worklist Application. By default, the class path to the resource bundle is:

```
oracle.bpel.worklistapp.resource.WorklistResourceBundle
```

Figure 32-41 shows the Application Preferences page with the **Resource Bundle** field highlighted. You reach the Application Preferences page by clicking **Administration** on the global toolbar at the very top of the Worklist Application interface.

Figure 32-41 Specifying the Resource Bundle



As an administrator, you can add or modify strings shown in the application by creating a custom resource bundle. You can then use the **Resource Bundle** field in the Application Preferences page to specify the class path to your custom resource bundle.

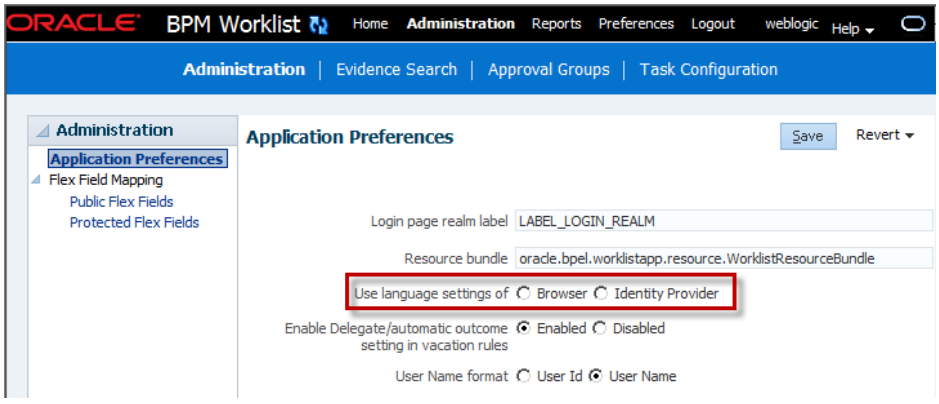
For more information about customizing resource bundles, see *Managing and Monitoring Processes with Oracle Business Process Management*.

How to Specify the Language Locale Information

From the Application Preferences page, you can specify how the Worklist Application display language is determined. Information about the language locale can be derived from either the user's browser or the identity provider that stores information on worklist users.

Figure 32-42 shows the Applications Preferences page with the Use language settings of options highlighted. You reach the Application Preferences page by clicking **Administration** on the global toolbar at the very top of the Worklist Application interface.

Figure 32-42 Specifying Language Local Information



How to Specify User Name Format

From the Application Preferences page, you can specify how the user's name is displayed on the screen after they have logged in. You can choose to display the userid, such as jstein, or the user's name, John Steinbeck.

How to Specify a Branding Logo

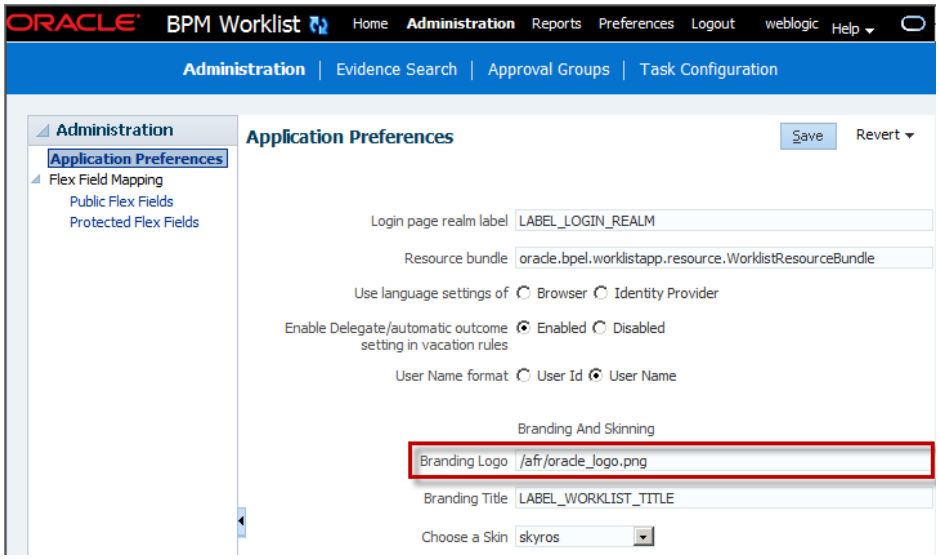
A branding logo is the image displayed in the top left corner of every page of the Worklist Application. The Oracle logo is the default, and you can change it to one of your choosing.

Note:

The ideal image size is 120px x 40px (length x width) for proper display. Although images with high resolution and size are compressed to fit the branding logo size, smaller images display better.

Figure 32-43 shows the Application Preferences page with the **Branding Logo** field highlighted. You reach the Application Preferences page by clicking **Administration** on the global toolbar at the very top of the Worklist Application interface.

Figure 32-43 Specifying the Branding Logo



To specify the branding logo:

Do one of the following:

- Refer to an external image-hosting web site. To do this task: In the **Branding Logo** field, enter the URL of the image.
- Upload an image to a particular location on the server and, in the **Branding Logo** field, enter its relative path, for example, `/afr/my_logo.png`.
- Refer to an image from the shared library. To do this task: In the **Branding Logo** field, enter the path of the logo name as found in the shared library, for example, `/my_logo.pngv`.

Note:

Customizing the branding logo from either the Worklist Application or Process Workspace changes the logo in both applications. For example, if you change the logo from Worklist, the Workspace logo is changed automatically.

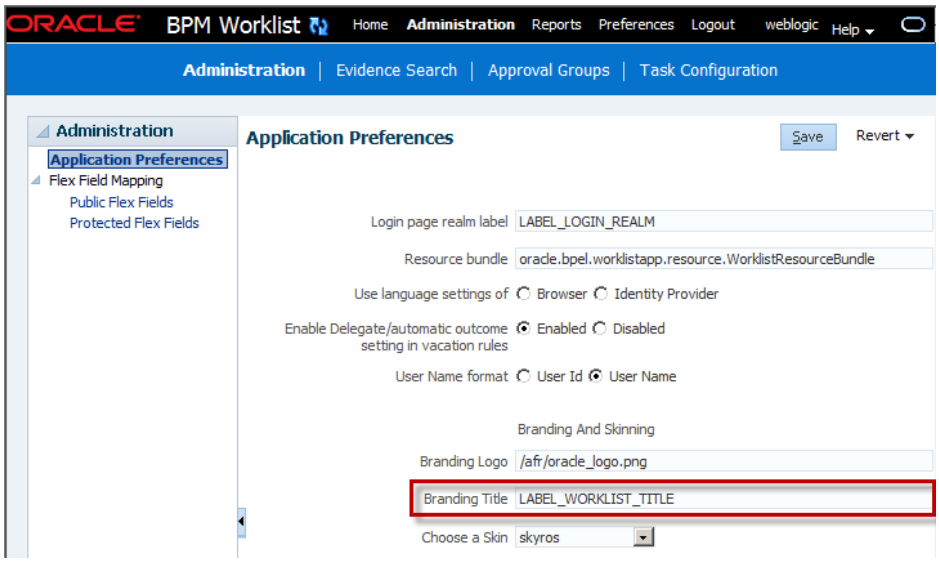
For information about deploying images and JAR files as part of a shared library, see *Managing and Monitoring Processes with Oracle Business Process Management*.

How to Specify the Branding Title

You can specify the title for your site, changing the default title, *BPM Worklist*, to one that you choose.

Figure 32-44 shows the Application Preferences field with the **Branding Title** field highlighted. You reach the Application Preferences page by clicking **Administration** on the global toolbar at the very top of the Worklist Application interface.

Figure 32-44 Specifying the Branding Title



To specify the branding title:

Do one of the following:

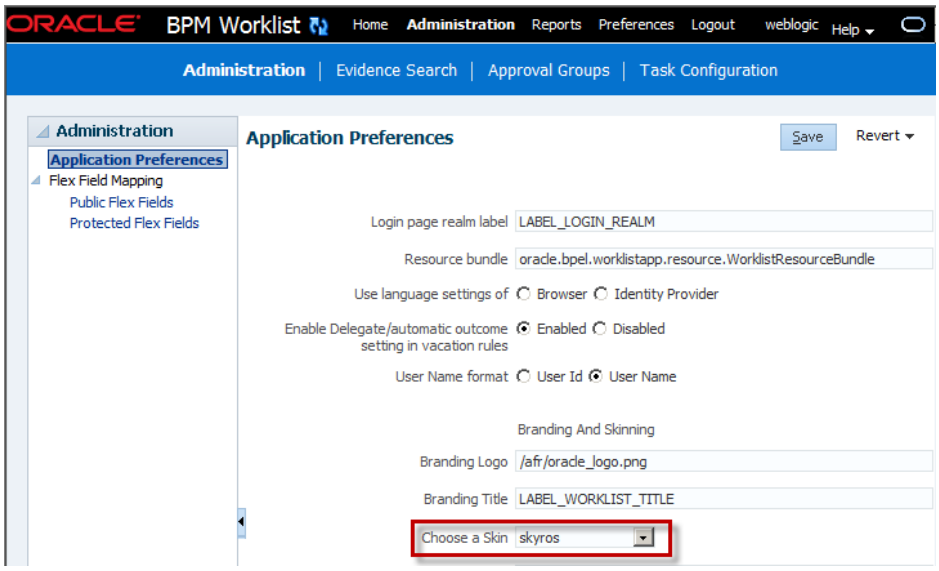
- In the **Branding Title** field, enter a simple string for your title.
- In the **Branding Title** field, enter a label that refers to a key-value pair in the Resource Bundle. In this way, you can internationalize your title, for example, LABEL_WORKLIST_TITLE.

How to Choose a Skin

A skin determines the look and feel of your graphical interface. You specify the skin from the Application Preferences page. You reach the Application Preferences page by clicking **Administration** on the global toolbar at the very top of the Worklist Application interface.

Figure 32-45 shows the Application Preferences page with the Choose a Skin field highlighted.

Figure 32-45 Choosing a Skin



To Choose A Skin

To choose a skin:

Do one of the following:

- From the **Choose a Skin** list, select one of the default ADF skins
- Upload your own customized skin `.css` file in a `.JAR` file and deploy it as a part of shared library. Then, when you restart your application from the console, your custom skin appears in the **Choose a Skin** list.

To Create a JAR File Containing Customized Skins

To create a JAR file containing customized skins:

1. Create a directory structure similar to the following example:

```
C:\temp\META-INF\adf\oracle\skin\images
     META-INF\skins\custom.css
     META-INF\trinidad-skins.xml
```

   In this example, you can change the word *`custom`* to the name of your own customized skin.

2. Make sure the content of `trinidad-skins.xml` file is as follows:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<skins xmlns="http://myfaces.apache.org/trinidad/skin">
   <skin>
     <id>custom.desktop</id>
     <family>custom</family>
     <extends>custom.desktop</extends>
     <render-kit-id>org.apache.myfaces.trinidad.desktop</render-kit-id>
```

```
      <style-sheet-name>skins/custom.css</style-sheet-name>
    </skin>
</skins>
```

3. Create the .JAR file by issuing the following command from the `c:\temp` directory:

   ```
   jar -cvf customSkin.jar META-INF/
   ```

4. Copy this JAR file to the directory `/scratch/`*`username`*`/sharedLib`.

Note:

Refer to the images in your css file this way:

`../adf/oracle/skin/images/example.gif` (with the two trailing dots).

This allows the search for the META-INF root to start one directory above the `META-INF/skin` directory in which the `.css` file is located.

For information about deploying JAR files as part of a shared library, see *Managing and Monitoring Processes with Oracle Business Process Management*.

How to Enable Customized Applications and Links

For Process Workspace, you can create customized external applications and links that become available in the External Applications panel. Moreover, in both Process Workspace and the Worklist Application, you can specify the columns that appear in the Task Details pane.

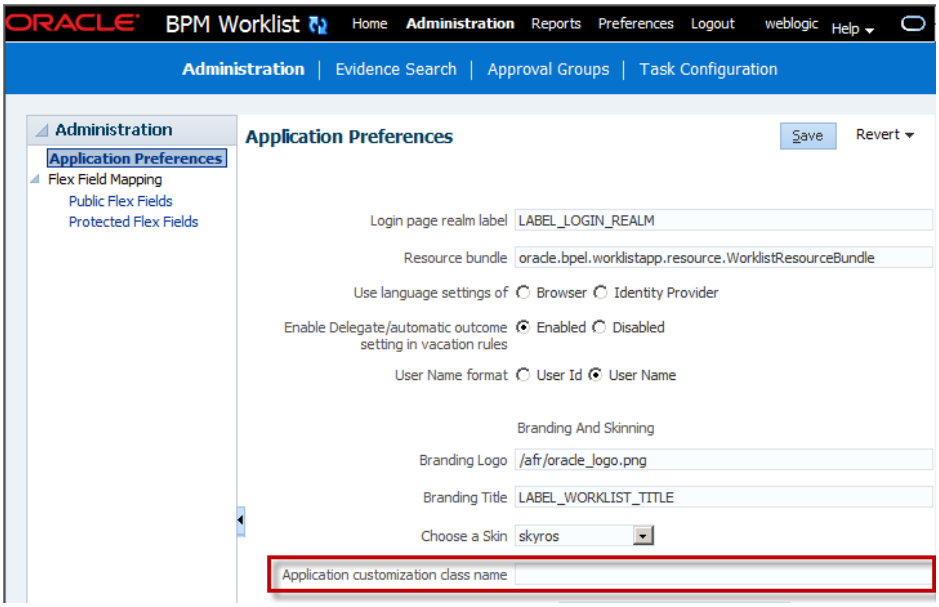
You specify a custom application by using the Application Preferences page. You reach the Application Preferences page by clicking **Administration** on the global toolbar at the very top of the Worklist Application interface.

To see the Java code for specifying a custom application, see Java Code for Enabling Customized Applications in .

To enable customized applications:

1. In the Application Preferences page, enter the class name of your custom application in the **Application customization class name** field, as shown in Figure 32-46.

Figure 32-46 Specifying a Custom Application



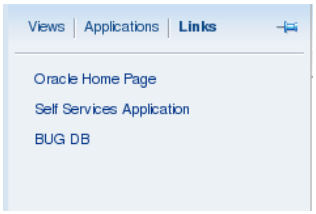
2. Restart the application from the console.

   Depending on your customization, you can now see its effects.

   If your customization is for Process Workspace and involves either creating an external application or specifying inbox columns in the Task Details pane or both, you see the following:

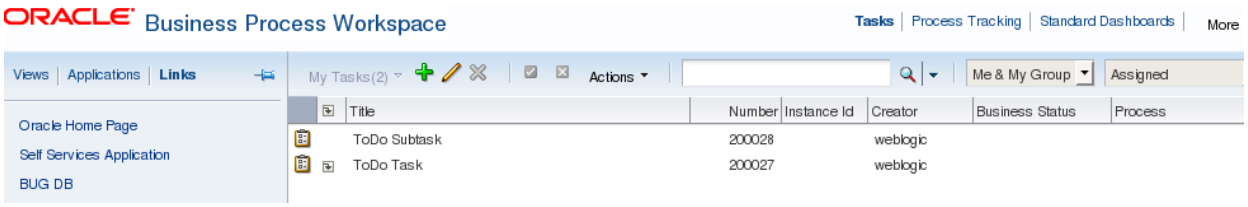
   - Your custom application listed in the External Applications panel of the Process Workspace Home page as shown in Figure 32-47.

     Figure 32-47 External Applications Panel in Process Workspace

   - The columns of the Task Details inbox adjusted according to your specifications as shown in Figure 32-48.

Figure 32-48 Customized Columns in Task Details Pane



For more information about customizing applications in Worklist Application and Process Workspace, see *Managing and Monitoring Processes with Oracle Business Process Management*.

How to Specify an Image for a Task Action

If you are an administrator, then you can specify whether an action is displayed with a red X icon or with a green check mark icon.

To specify an image for a task action:

1. Select **Administration**, then **Application Preferences**.
2. From the lists in the **Map task actions to an image** field, select the tasks you want to map to either the green check mark icon or the red X icon.
3. Click **Save**.

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.

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.

Data Types

Table 32-9 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 32-9 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.

Attributes

Table 32-10 lists the predefined attributes for messaging filters.

Table 32-10 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
Customer Name	String
Expiration Date	Date
Order Type	String
Organization	String
Classification	String
Service Request Type	String

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.

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 **Available Channels** to view, create, edit, and delete messaging channels.

Viewing Your Messaging Channels

You can display your existing messaging channels.

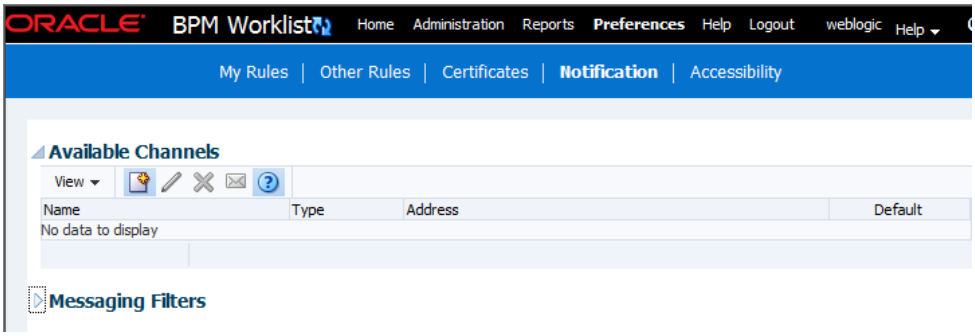
To view messaging channels:

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

The Available Channels list appears (Figure 32-49) 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 32-49 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.

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.

Managing Messaging Filters

You can use **Messaging Filters** 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.

Viewing Messaging Filters

You can display your existing messaging filters.

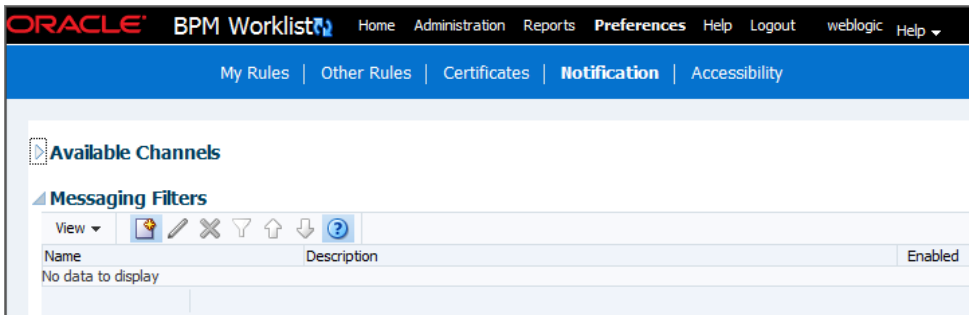
To view your messaging filters:

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

   The Messaging Filters list appears (Figure 32-50) and displays the following information:

   - **Name**: The name of the messaging filter
   - **Description**: An optional description of the messaging filter
   - **Enabled**: Specifies if this filter is being used in message handling

Figure 32-50 Messaging Filters



3. Click **View > Columns** and select the columns to display or hide.

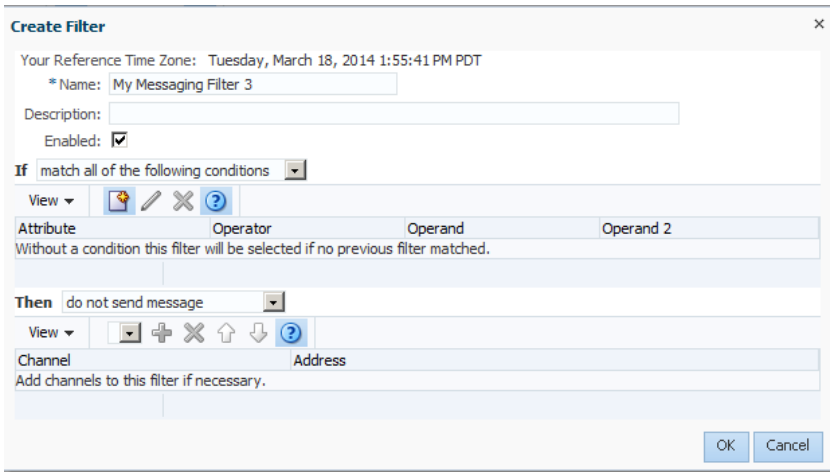
Creating Messaging Filters

To create a messaging filter:

1. Click **Create**.

   The Create Filter dialog box appears, as shown in Figure 32-51.

Figure 32-51 Adding a Messaging Filter



2. Specify the following information:

   - **Name**: The name of the messaging filter.
   - **Description**: An optional description for the messaging filter.
   - **Enabled**: By default this option is checked. Clear if you do not want this filter used in message handling.

3. Select whether notifications must meet all of the conditions or any of the conditions by selecting either the **Match all of the following conditions** or the **Match any of the following conditions** options.

4. Click **Create**.

   Define the filter conditions in the **Create Condition** dialog box, as follows:

   a. Select the attribute from the list.

   b. Select the operator, such as **isEqual**, from the list.

   c. Type the value of the condition in the **Operand** field.

   d. Click **OK** to add the condition to the list.

   e. Repeat these steps to add more filter conditions. To remove a filter condition, click **Delete**.

5. Select from the following messaging options in the **Action** section:

   - **Do not send messages**: Do not send a message to any channel.
   - **Send to all selected channels**: Send a message to all specified channels in the address list.
   - **Send to 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.

6. To set the delivery channel, select a channel from the **Add Notification Channel** list and click **Add**. To remove a channel, click **Delete**.

7. 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*.

8. Click **OK**.

   The messaging filter appears on under Messaging Filters area. The Messaging Filters area enables you to edit or delete the channel.

Editing a Messaging Filter

To edit a messaging filter:

1. Select the filter from the Messaging Filters area.
2. Click **Edit**.
3. Click **OK** to update the messaging filter. Click **Cancel** to dismiss the dialog without modifying the filter.

Deleting a Messaging Filter

To delete a messaging filter:

1. Select the filter from the Messaging Filters area.
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.

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:

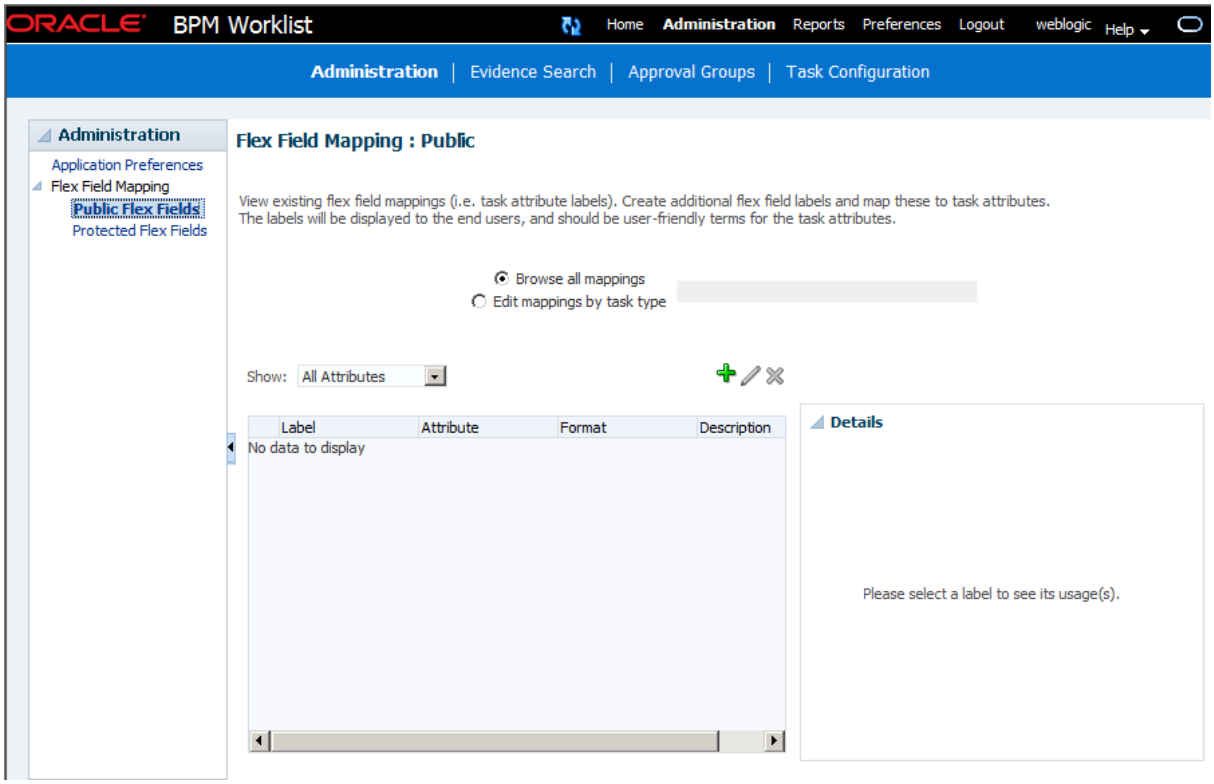
- Mapped attributes must be defined before instances of the business process are generated. Only instances generated after mapped attributes are created reflect the correct mapped attributes. Older instances of the business process do not reflect subsequent mapped attribute changes.
- When you add a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

How To Map Attributes

An administrator, or users with special privileges, can use attribute mapping, shown in Figure 32-52, 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 32-52 Mapped Attribute Mapping

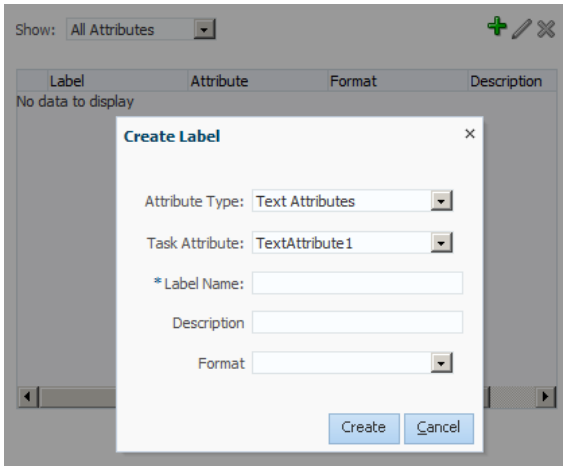


To Create Labels

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 32-53.

Figure 32-53 Creating a Label



As Figure 32-53 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:

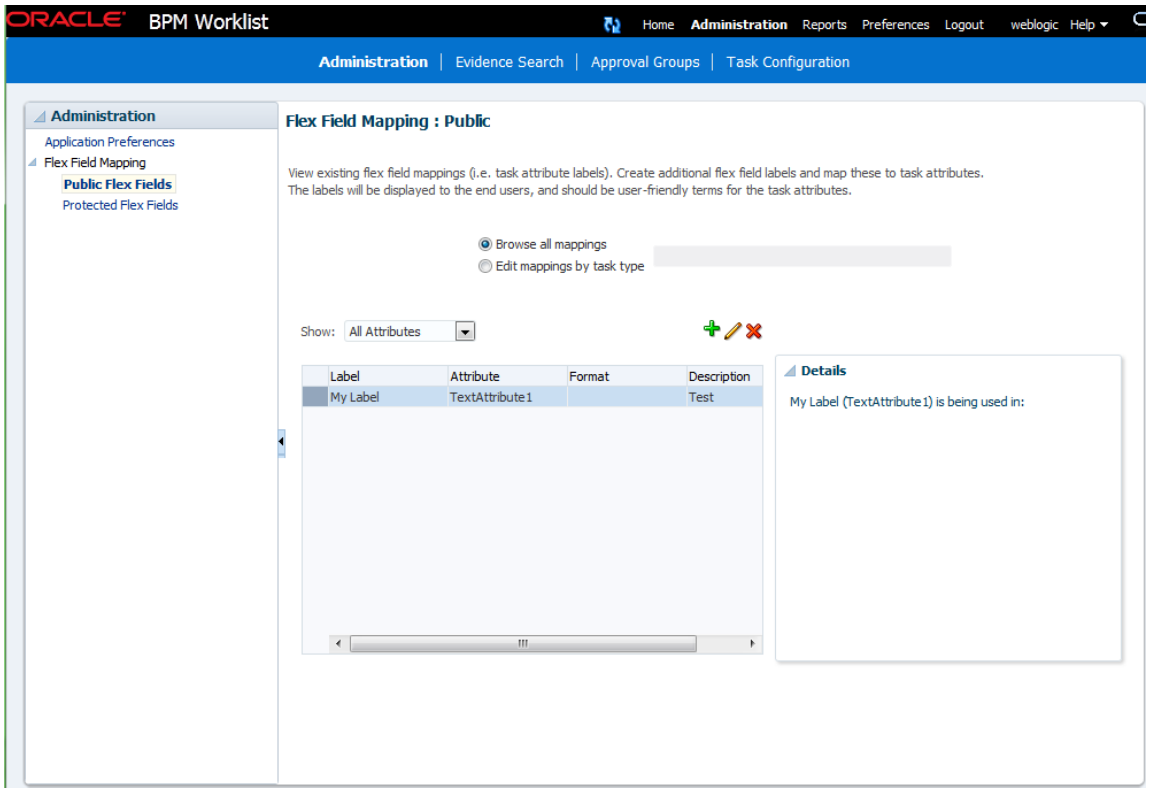
- Only simple type payload attributes can be mapped.
- A mapped attribute (and thus a label) can be used only once per task type.
- Data type conversion is not supported for the `number` or `date` data types. For example, you may not map a payload attribute of type `string` to a label of type `number`.

To Browse All Mappings

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 32-54 Browsing Mappings

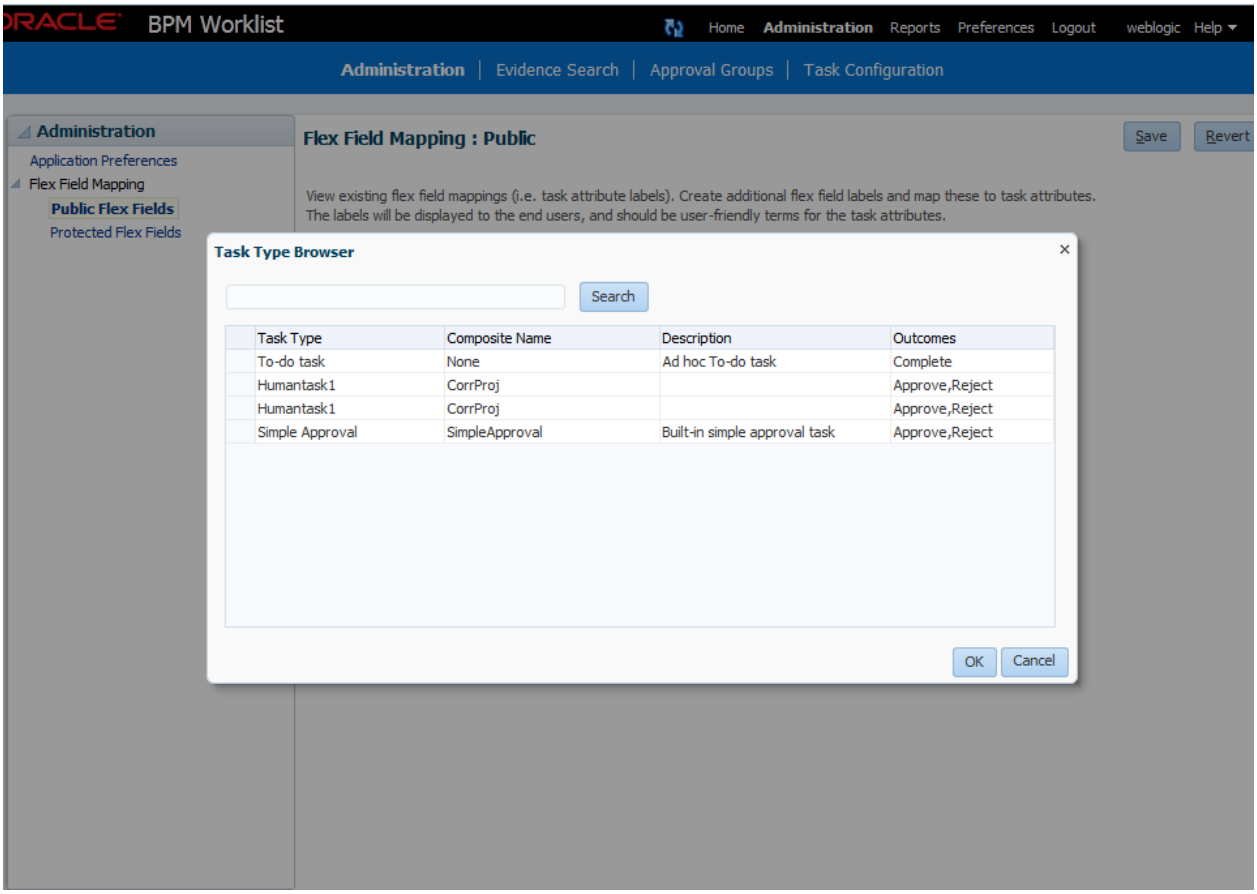


To Edit Mappings by Task Type

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**.

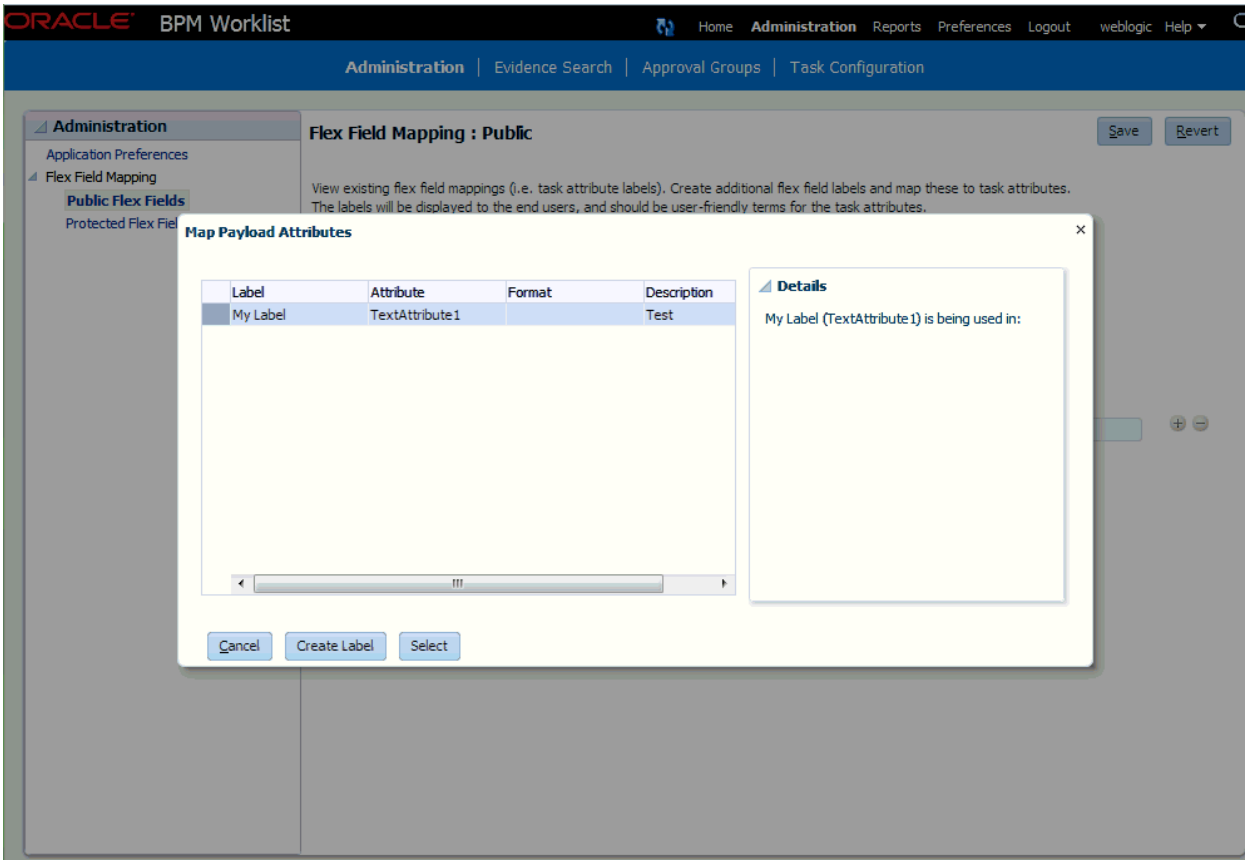
Figure 32-55 Selecting a Task Type



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 32-56.

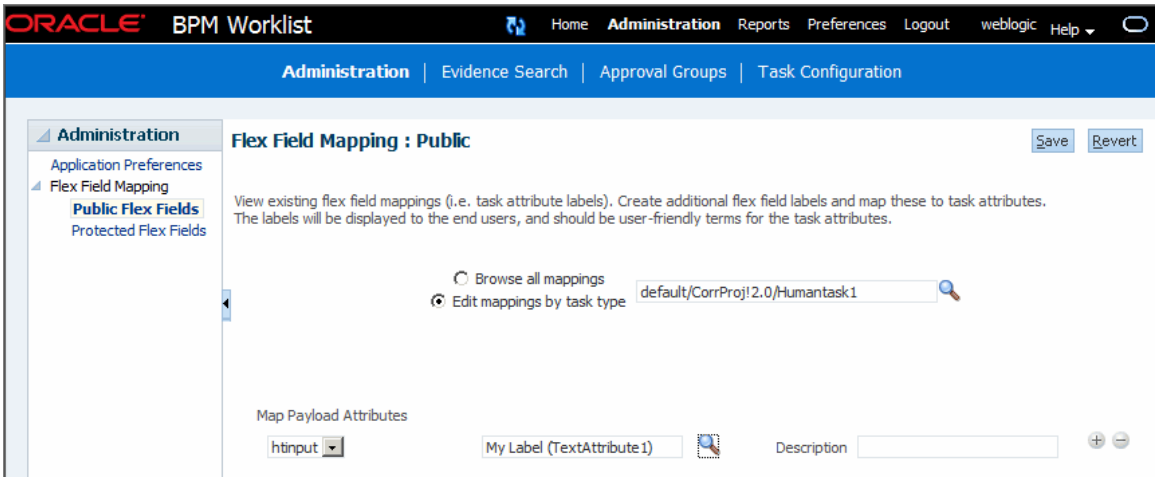
Figure 32-56 Selecting a Label



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

   Figure 32-57 shows a completed mapping.

Figure 32-57 Mapped Attribute Mapping Created



See Internationalization of Attribute Labels for more information.

Custom Mapped Attributes

The following mapped attributes are included in the `WorkflowTask.xsd` file and are available for your use without restrictions.

Table 32-11 Custom Mapped Attributes

Attribute	Data Type
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`.

Creating Worklist Reports

Table 32-12 lists the worklist reports available for task analysis.

Table 32-12 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.

Table 32-12 (Cont.) Worklist Report Types

Report Name	Description	Input Parameters
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).

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.

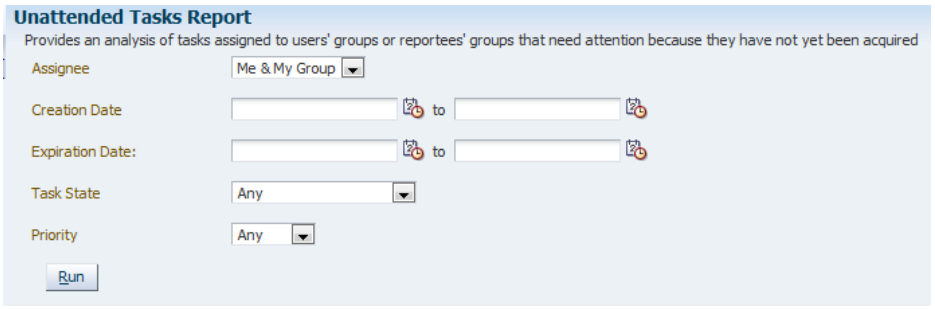
   Figure 32-58 shows the report types available.

Figure 32-58 Oracle BPM Worklist Reports

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

   Figure 32-59 shows an example of the Unattended Tasks Report input page. The other reports are similar. See Table 32-12 for information about input parameters for all the report types.

Figure 32-59 Unattended Tasks Report—Input Page for Task Analysis

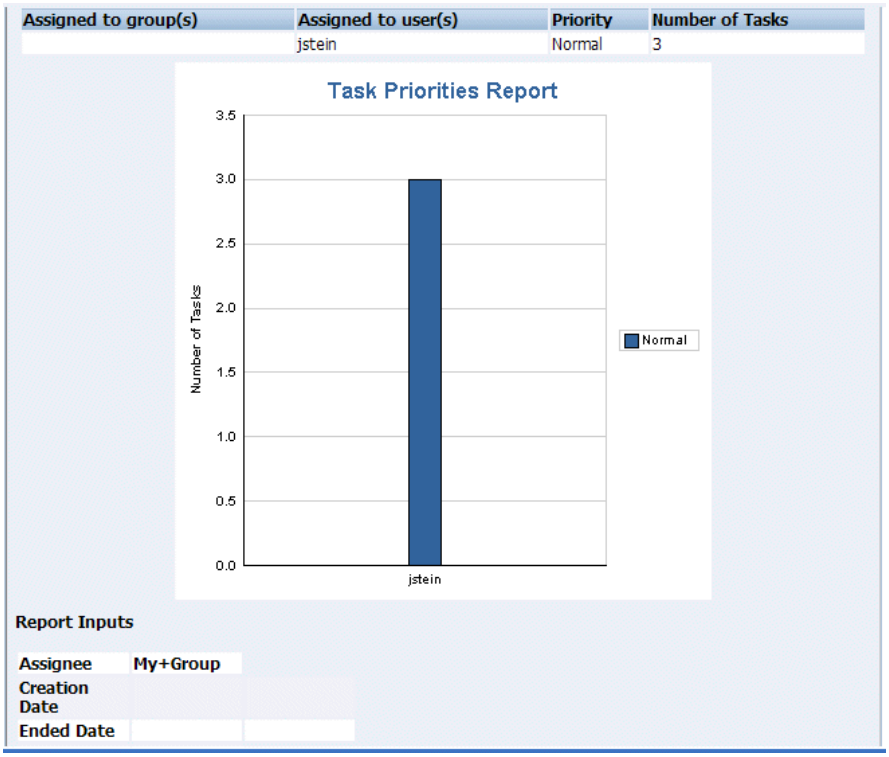


4. Click **Run**.

What Happens When You Create Reports

As shown in Figure 32-60, 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).

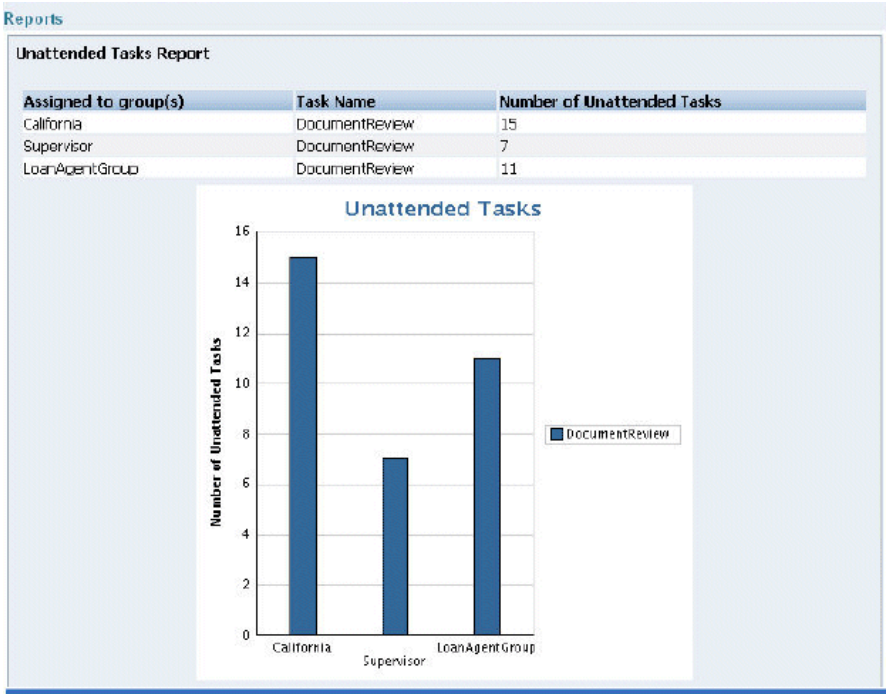
Figure 32-60 Report Display—Table Format, Bar Chart Format, and Report Inputs



Unattended Tasks Report

Figure 32-61 shows an example of an Unattended Tasks report.

Figure 32-61 Unattended Tasks Report



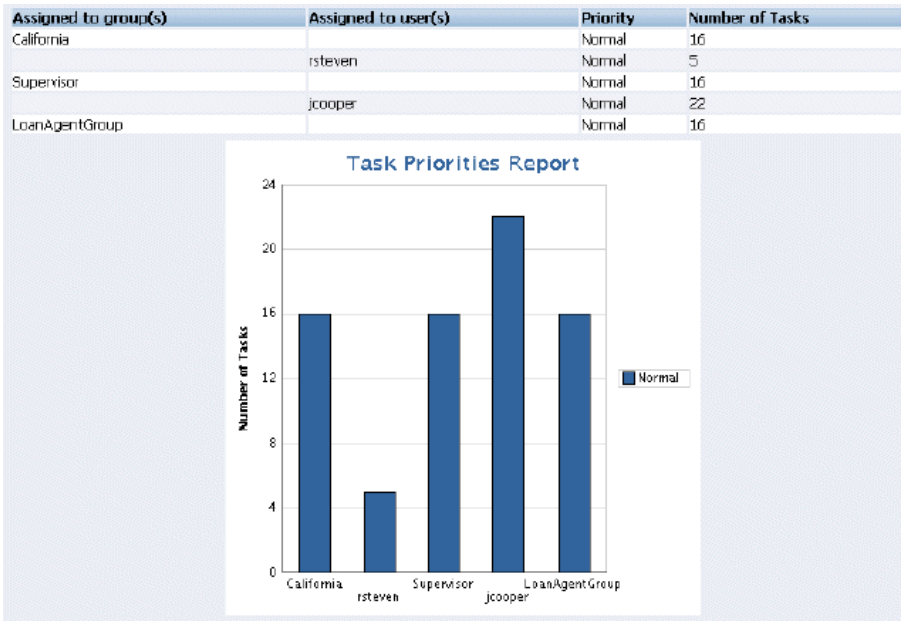
The report shows that the California group has 15 unattended tasks, the Supervisor group has 7 unattended tasks, and the LoanAgentGroup has 11 unattended tasks. The

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.

Tasks Priority Report

Figure 32-62 shows an example of a Tasks Priority report.

Figure 32-62 Tasks Priority Report

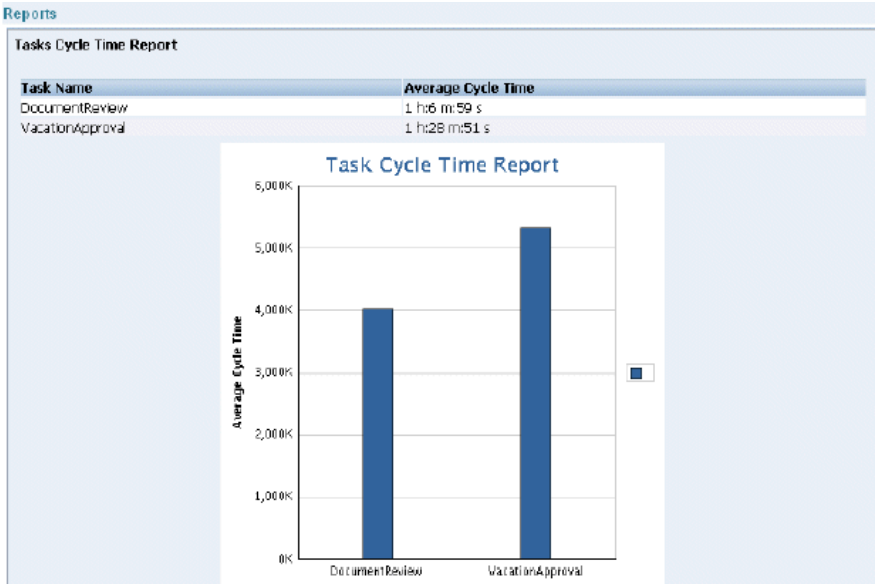


The report shows that the California group, the Supervisor group, and the LoanAgentGroup each have 16 tasks of normal priority. The users rsteven and jcooper have 5 and 22 tasks, respectively, all normal priority. Priorities (highest, high, normal, low, lowest) are distinguished by different colors in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

Tasks Cycle Time Report

Figure 32-63 shows an example of a Tasks Cycle Time Report.

Figure 32-63 Tasks Cycle Time Report

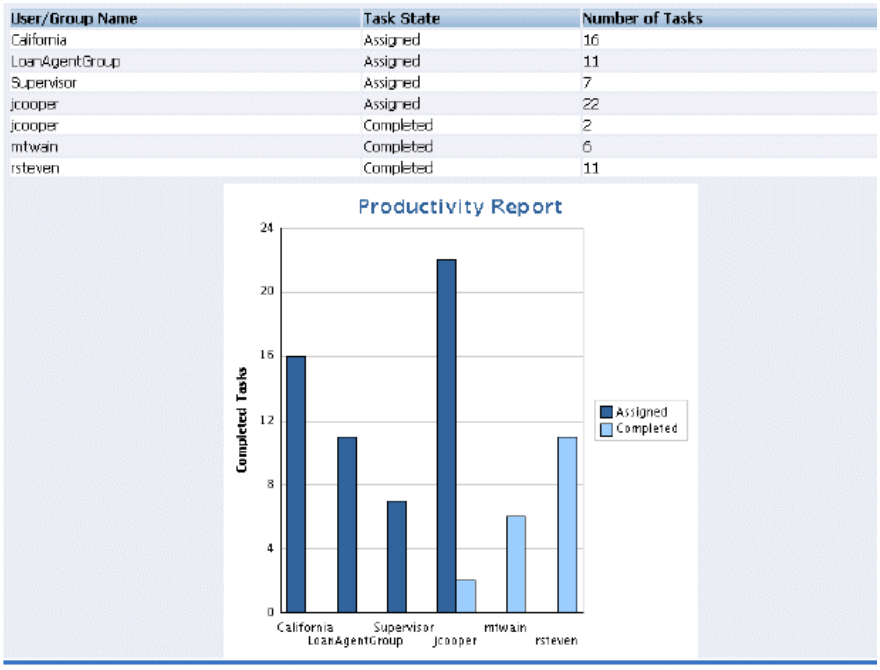


The report shows that it takes 1 hour and 6 minutes on average to complete DocumentReview tasks, and 1 hour and 28 minutes on average to complete VacationApproval tasks. The bar chart shows the average cycle time in milliseconds.

Tasks Productivity Report

Figure 32-64 shows an example of a Tasks Productivity Report.

Figure 32-64 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.

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 the 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:

- How to Specify the Login Page Realm Label for how to select **Browser** or **Identity Provider** from the Application Preferences page
- Customizing the Task List Page and Figure 32-14

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 How to Specify the Resource Bundle) or by providing an updated version of the default bundle class. See the Workflow Customizations sample for details.

For task attribute names, mapped attribute labels, and dynamic assignment function names, the strings come from configuring the resource property file `WorkflowLabels.properties`. This file exists in the `wfresource` subdirectory of the services config directory. See 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 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.

Note:

You cannot use Korean characters in the human task name. In place of Korean characters, Oracle recommends using only letters A-Z, a-z, 0-9, and "_" in the human task name.

How to Change the Preferred Language, Display Names of Users, and Time Zone Settings 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, the 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, "Managing the Embedded LDAP Server" of *Administering Security for Oracle WebLogic Server*.
   - For Oracle Internet Directory:
     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 either add or set the `preferredLanguage` attribute. See Table 32-13 for a list of supported languages.

   You can also determine the language in which user names are displayed. To do this task, navigate to the user entry in the LDAP directory, then add or specify the `displayname` attribute.

Note:

- The user name that appears in the Assignee column in the worklist does not honor the setting of the `displayname` attribute.
- Display names are taken from LDAP. So even when you change the display name, only the LDAP user name is displayed when you log into workspace.

To change the time zone setting, either add or 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. 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.

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 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**.

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</preferredLanguage>
```

Oracle BPM Worklist supports the languages shown in Table 32-13.

Table 32-13 Languages Supported in Oracle BPM Worklist

Language	Format
English	(en)
French	(fr)
German	(de)
Spanish (International)	(es)
Italian	(it)
Portuguese (Brazil)	(pt-BR)
Japanese	(ja)
Korean	(ko)
Chinese (Traditional)	(zh-TW)
Chinese (Simplified)	(zh-CN)
Arabic	(ar)
Czech	(cs)
Danish	(da)
Dutch	(nl)
Finnish	(fi)
Greek	(el)
Hebrew	(he)

Table 32-13 (Cont.) Languages Supported in Oracle BPM Worklist

Language	Format
Hungarian	(hu)
Norwegian	(no)
Polish	(po)
Portuguese	(pt)
Romanian	(ro)
Russian	(ru)
Slovak	(sk)
Swedish	(sv)
Thai	(th)
Turkish	(tr)
Canadian French	(fr-CA)

What You May Need to Know Setting Display Languages in Worklist

Oracle BPM Worklist can be configured to set the language from the browser or from the identity store (LDAP). There are two levels to this setting: the application level and the user level. If the user preference is set, as LDAP in the user setting, it takes precedence in determining the worklist display language. If you do not set a language in LDAP, worklist follows default language as server locale. However, email notifications always follow the language set in LDAP. If no language is set in LDAP, email notifications follow server locale.

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. Some regions include sub-regions, for example `America/Indiana/Indianapolis`.

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 you 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.

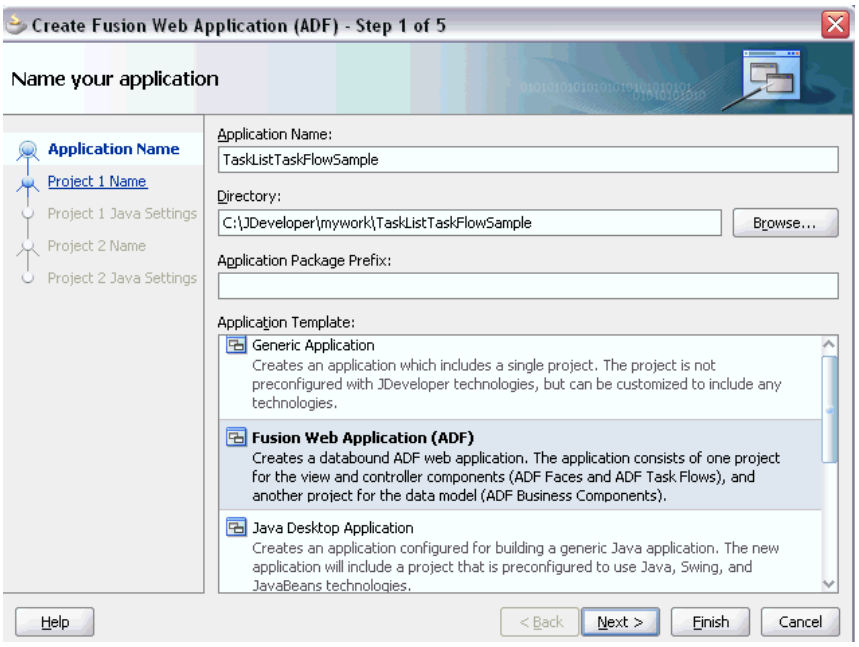
How to Create an Application With an Embedded Reusable Worklist Region

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 32-65 provides details.

Figure 32-65 Creation of Application with an Embedded Reusable Worklist Region

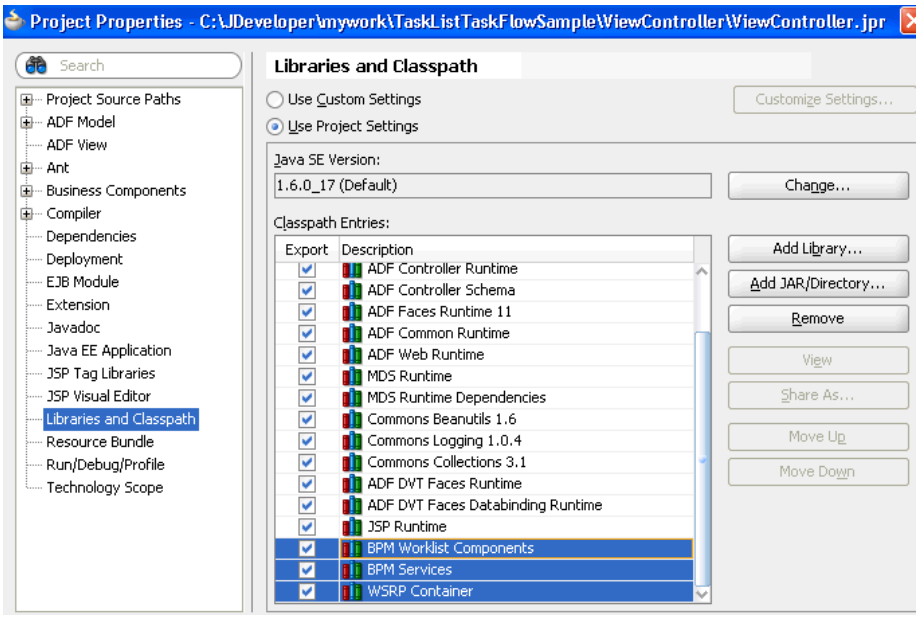


2. Open the View Controller Project Properties, **Libraries and Classpath** section, and click **Add Library** to add the following libraries in the class path:

   - **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 32-66 provides details.

Figure 32-66 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.

      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.

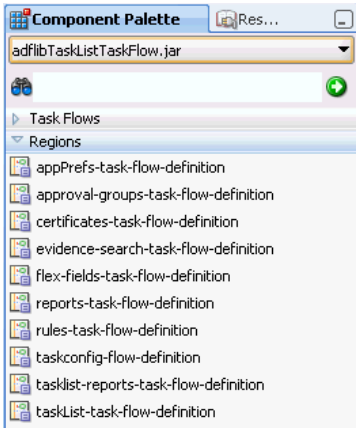
      If you run the Task List task flow in federated mode, you do not need to do this step. See "federatedMode" in section What You May Need to Know About Task List Task Flow for information about how to use the task flow in federated mode.

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 Components window. It contains the list of all the Task Flows and Regions. Figure 32-67 provides details.

Figure 32-67 Components Window



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:

   - What You May Need to Know About Task List Task Flow
   - What You May Need to Know About Certificates Task Flow
   - What You May Need to Know About the Reports Task Flow
   - What You May Need to Know About Application Preferences Task Flow
   - What You May Need to Know About Mapped Attributes Task Flow
   - What You May Need to Know About Rules Task Flow
   - What You May Need to Know About Approval Groups Task Flow
   - What You May Need to Know About Task Configuration 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="taskFlowMode" value="MODE_WORKLIST"/>
      <parameter id="showTaskDetailsPanel" value="true"/>
      <parameter id="showActionDropdown" value="true"/>
      <parameter id="showViewFilter" value="true"/>
      <parameter id="showStatusFilter" value="true"/>
      <parameter id="showSearchControl" value="true"/>
   </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>
```

If the generated custom application is a module, use `weblogic.xml`.

```
<library-ref>
        <library-name>oracle.soa.worklist.webapp</library-name>
</library-ref>
```

Before deploying the application, see How to Set Up the Deployment Profile.

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`.

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" in section What You May Need to Know About Task List Task Flow 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.

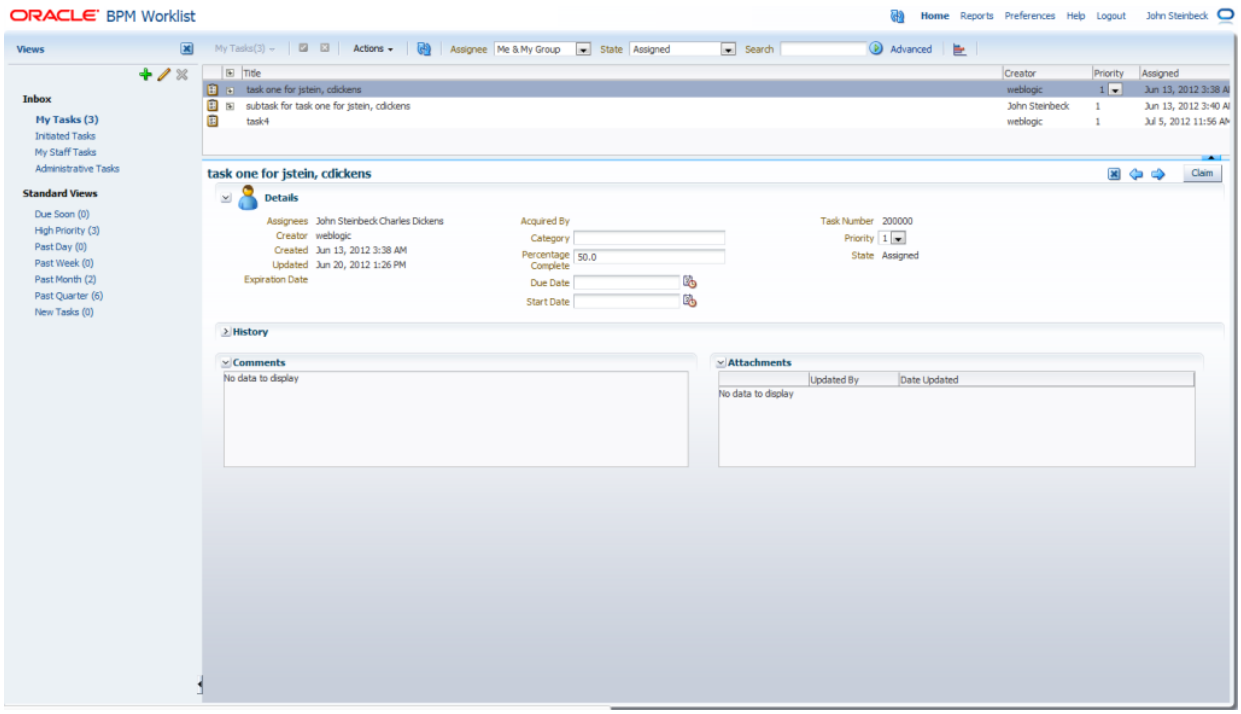
To restart the servers:

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.

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 32-68 provides details.

Figure 32-68 Task List



Some of the parameters are listed below.

- 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.
- 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" below 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 sample below for details about constructing the JAXB object (`WorkflowServicesClientConfigurationType`).

Make sure that you set one of the servers as `default`, as shown in the code sample below. 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.

```
import oracle.bpel.services.workflow.client.config.IdentityPropagationType;
import oracle.bpel.services.workflow.client.config.PolicyReferenceType;
import oracle.bpel.services.workflow.client.config.PolicyReferencesType;
import oracle.bpel.services.workflow.client.config.RemoteClientType;
import oracle.bpel.services.workflow.client.config.ServerType;
import oracle.bpel.services.workflow.client.config.SoapClientType;
import
oracle.bpel.services.workflow.client.config.WorkflowServicesClientConfigurationType;

WorkflowServicesClientConfigurationType wscct =
   new WorkflowServicesClientConfigurationType();

List<ServerType> servers = wscct.getServer();

/**** Setting default server in the list ****/

ServerType defalutServer = new ServerType();
servers.add(defalutServer);

defalutServer.setDefault(true);
defalutServer.setExcludeFromFederatedList(true);
defalutServer.setName("default");

RemoteClientType rct = new RemoteClientType();
rct.setServerURL("t3://myhost.us.example.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.example.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://myhost.us.example.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://myhost.us.example.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.example.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.example.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 the code sample below:

```
IWorkflowContext wfCtx =
wfSvcClient.getTaskQueryService().authenticate(username,password,realm,null);
wfCtxID = wfCtx.getToken();
```

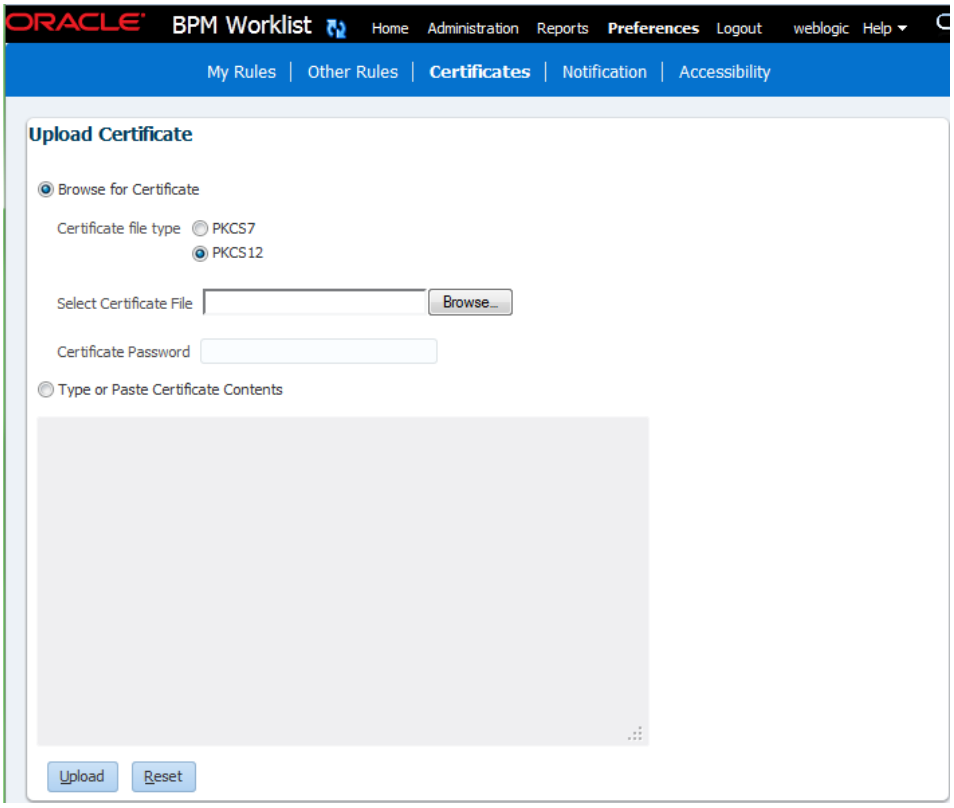
What You May Need to Know About Certificates Task Flow

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 32-69 provides details.

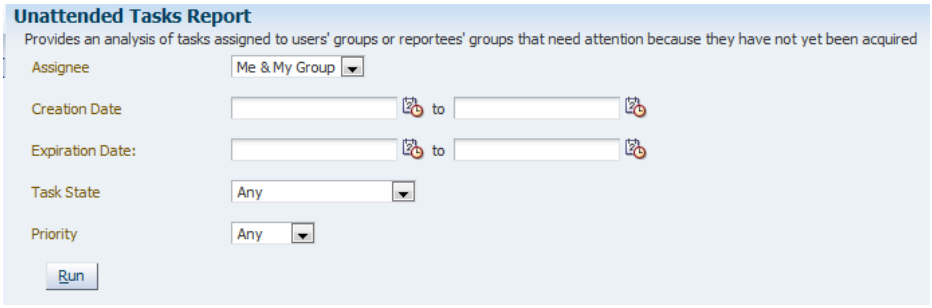
Figure 32-69 Digital Certificate



What You May Need to Know About the Reports Task Flow

Figure 32-70 shows the unattended tasks report.

Figure 32-70 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).

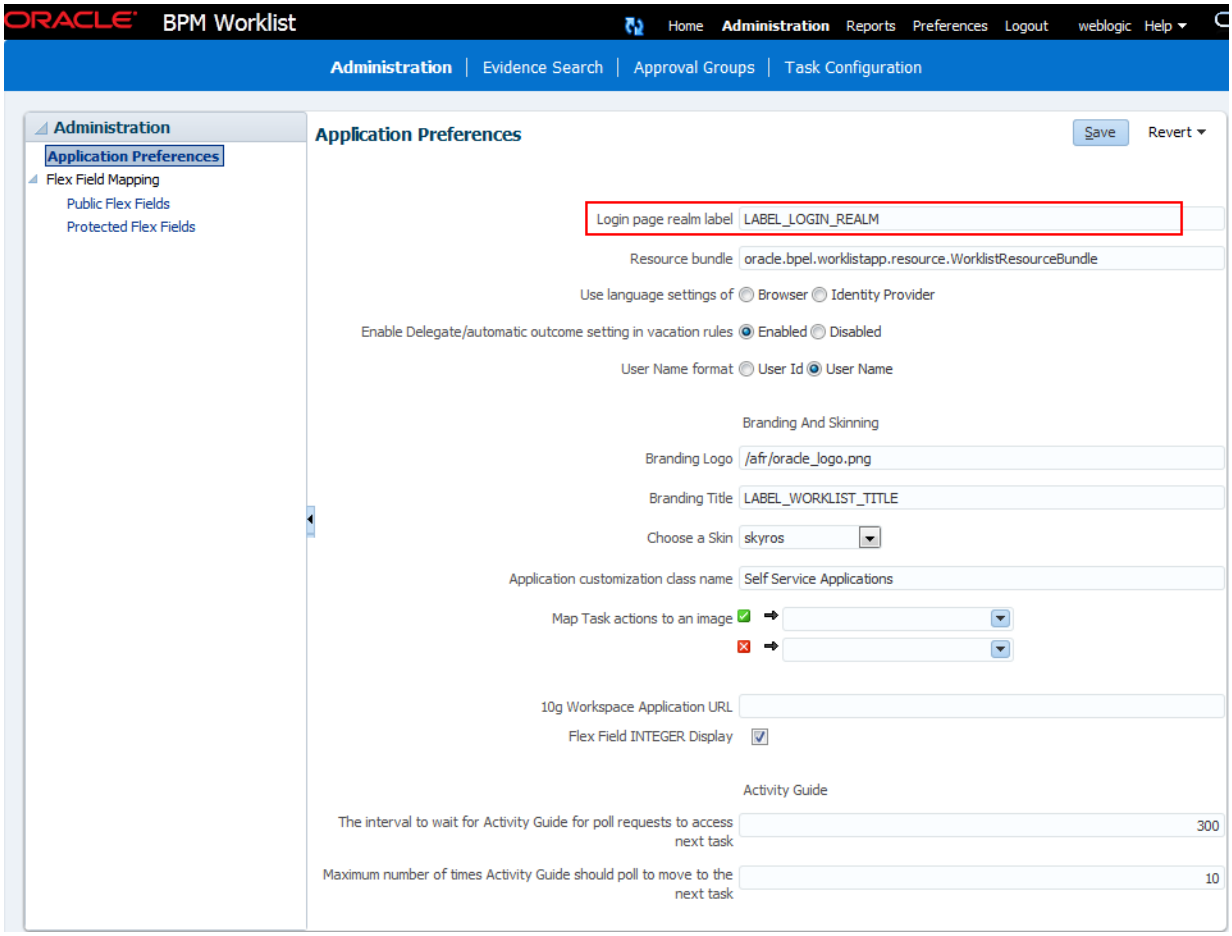
- **From...to (date range)** - An optional creation date range. The default is one week.
- **Task Type** - Use the Search (flashlight) icon to select from a list of task titles. All versions of a task are listed on the Select Workflow Task Type page (optional).

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 32-71 provides details.

Figure 32-71 Application Preferences



What You May Need to Know About Mapped Attributes Task Flow

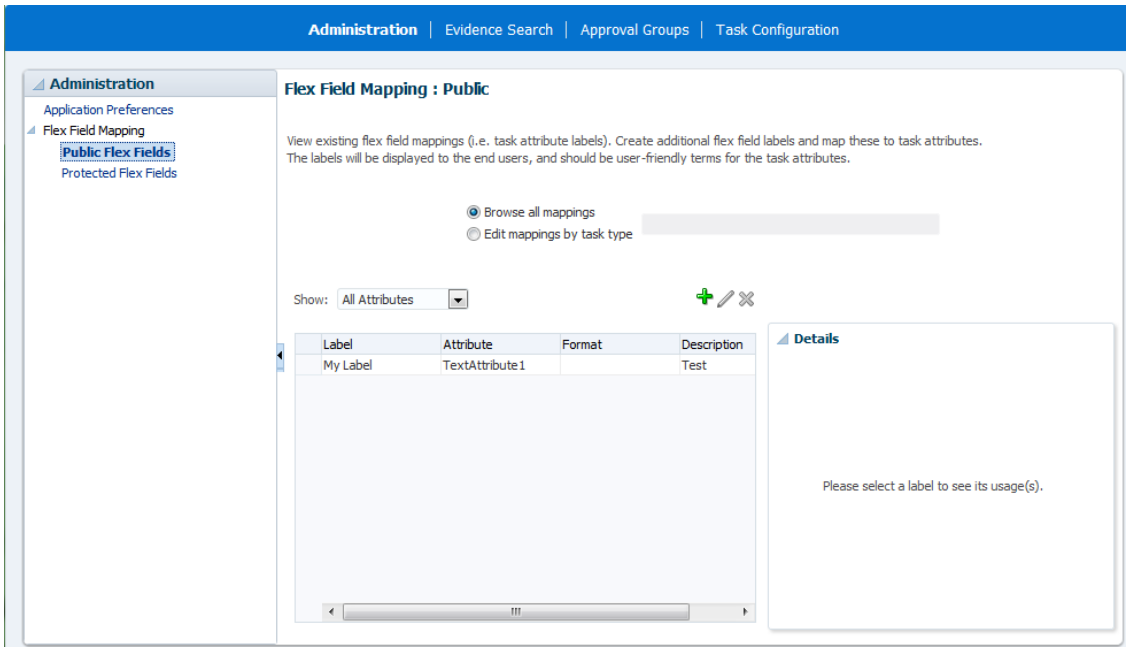
Human workflow mapped attributes store and query use case-specific custom attributes. These custom attributes typically come from the task payload values. Storing custom attributes in mapped attributes provides the following benefits:

- 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 32-72 provides details.

Figure 32-72 Mapped Attribute Mapping



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 32-73 and Figure 32-74 provide details.

Figure 32-73 Vacation Period

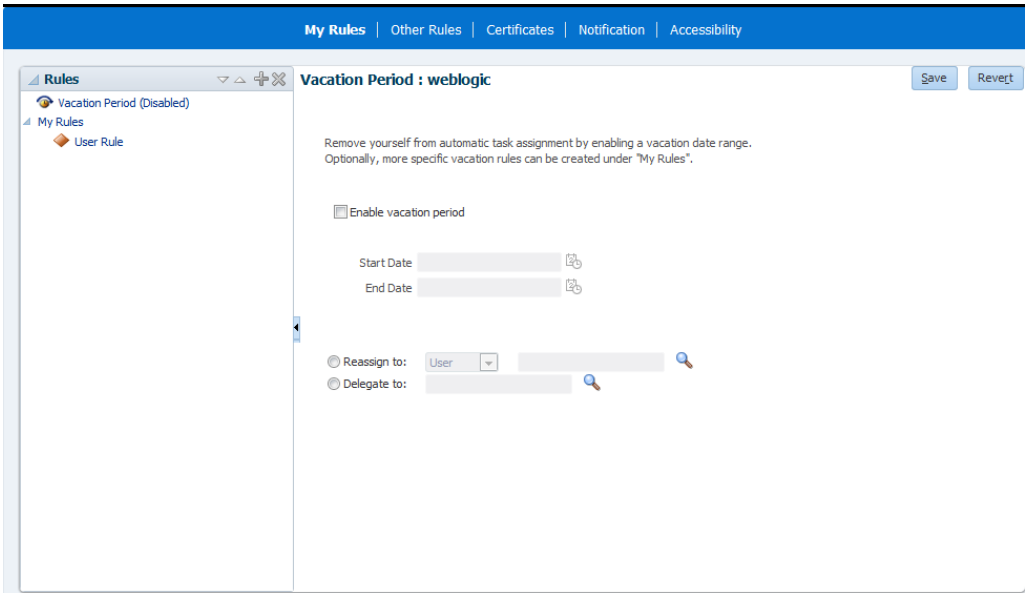
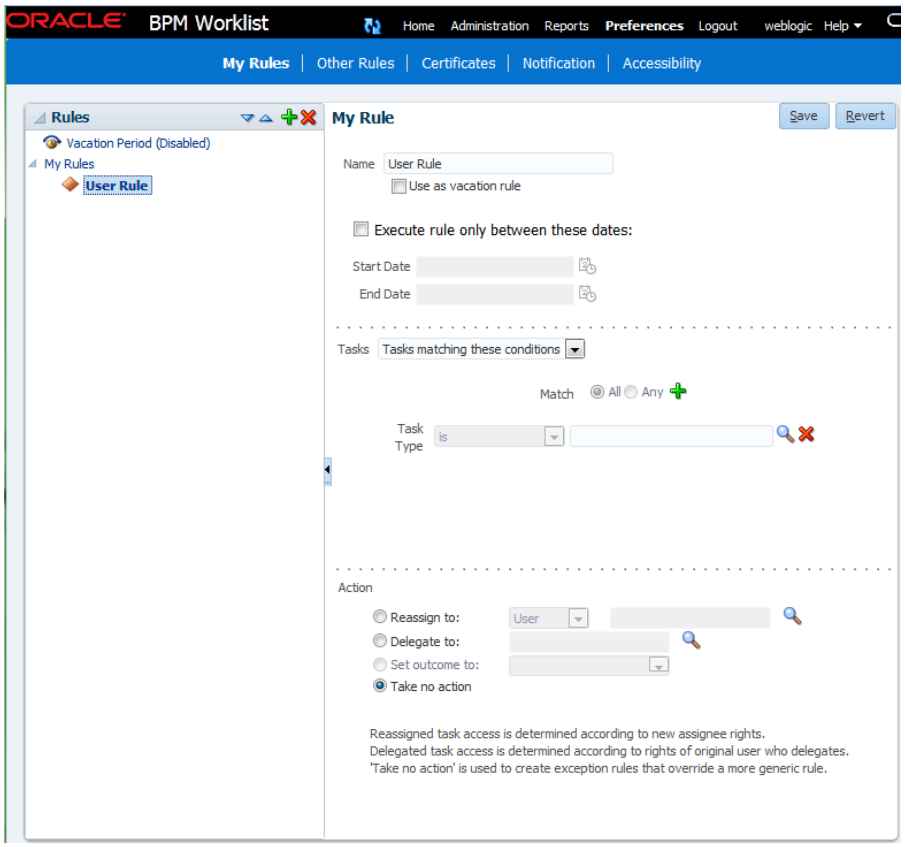


Figure 32-74 My Rule



What You May Need to Know About Approval Groups Task Flow

Approval groups are either a statically defined or a dynamically generated list of approvers. Approval groups usually are configured by the process owner using the worklist application. Typically, they are used to model subject matter experts outside

the transaction's managerial chain of authority, such as human resources or legal counsel, that must act on a task before or after management approval.

Static approval groups are predetermined lists of approvers, while dynamic approval groups generate approver lists at runtime. Dynamic approval groups require:

- delivery of an implementation according to the dynamic approver list interface by the developer
- registration of the implementation as a dynamic approval group using the Oracle BPM Worklist's UI by the IT department
- availability of the class file in a globally well-known directory that is part of the SOA class path

What You May Need to Know About Task Configuration Task Flow

Task Configuration is a web-based application in Worklist Application that enables business users and administrators to review and modify rules that were predefined by the workflow designer. These predefined rules can be changed for a specific customer based on the customer's applicable corporate policies.

For example, suppose that a corporate policy requires two levels of approvals for expense amounts greater than 1000. Suppose further that this policy is changed to require three levels. You can use Task Configuration to change the rule rather than having your IT department modify the rule in the underlying process and then deploy it again. Any change to the rule is applied starting with the next instance, and instances already in progress use the current rule definitions.

Task Configuration enables you to edit the event driven and data-driven rules associated with an approval flow at runtime—that is, when the workflow has already been deployed.

Java Code for Enabling Customized Applications in Worklist Application

How to Enable Customized Applications and Links explained how to specify a custom application by using the Application Preferences page of Worklist Application. The Java code for performing this specification is as follows:

```
package view.customisationimpl;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import oracle.bpel.services.workflow.client.IWorkflowServiceClient;
import oracle.bpel.services.workflow.runtimeconfig.IRuntimeConfigService;
import oracle.bpel.services.workflow.runtimeconfig.model.AttributeLabelType;
import oracle.bpel.services.workflow.runtimeconfig.model.AttributeLabelUsageList;
import oracle.bpel.services.workflow.runtimeconfig.model.AttributeLabelUsages;
import oracle.bpel.services.workflow.verification.IWorkflowContext;
import oracle.bpm.ui.customization.CustomLink;
import oracle.bpm.ui.customization.IBPMUICustomizations;

public class WorkspaceCustomisationImpl implements IBPMUICustomizations {
    private static Map displayNameMap = new HashMap();
    public WorkspaceCustomisationImpl() {
        displayNameMap.put("instanceId", "Instance Id");
        displayNameMap.put("protectedTextAttribute1", "Business Status");
    }
```

```
    public List<CustomLink> getCustomGlobalLinks() {
        CustomLink globalLink1 =
            new CustomLink("Oracle Home Page", "www.oracle.com", null);
        CustomLink globalLink2 =
            new CustomLink("Self Services Application", "http://global-
ebusiness.example.com/",
                           null);
        CustomLink globalLink3 =
            new CustomLink("BUG DB", "https://bug.example.com/", null);
        List<CustomLink> globalLinks = new ArrayList<CustomLink>();
        globalLinks.add(globalLink1);
        globalLinks.add(globalLink2);
        globalLinks.add(globalLink3);
        return globalLinks;
    }
    public String getColumnNames() {
        return "title,taskNumber,instanceId,creator,protectedTextAttribute1";
    }

    private static void initDisplayMap(IWorkflowServiceClient client,
        IWorkflowContext context) {
        // you can use service to load all label namess for text attributes
        if (displayNameMap == null) {
            synchronized (String.class) {
                if (displayNameMap == null) {
                    displayNameMap = new HashMap();
                    try {
                        IRuntimeConfigService service =
                            client.getRuntimeConfigService();
                        AttributeLabelUsageList list =
                            service.getAttributeLabelUsages(context, "Text");
                        List<AttributeLabelUsages> list1 =
                            list.getAttributeLabelUsages();
                        for (AttributeLabelUsages usage : list1) {
                            AttributeLabelType type = usage.getLabel();
                            displayNameMap.put(type.getTaskAttribute(),
                                               type.getLabelName());
                        }
                    } catch (Exception exc) {
                    }
                }
            }
        }
    }

    public String getColumnDisplayName(IWorkflowServiceClient client,
                                       IWorkflowContext context,
                                       java.lang.String colName) {
        initDisplayMap(client, context);
        return (String)displayNameMap.get(colName);
    }
}
```

33

Building a Custom Worklist Client

This chapter describes how, starting with the sample Worklist Application, a developer can build clients for workflow services by using the APIs exposed by the workflow service. The APIs enable clients to communicate with the workflow service by using remote EJBs, SOAP, and HTTP.

This chapter includes the following sections:

Introduction to Building Clients for Workflow Services

When creating a Java client application to call Human Workflow service, ensure that JRF is running on the same environment as the Java client application. 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`.

The code sample below 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.

```
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());
}
```

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.

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 1 and 2 below:

  Example 1

```
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 2

```
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. The code sample below shows how a logger can be passed 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 34-1 for the clients available for each of the services.

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 the code sample below:

```
<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 the code sample below:

```
<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.

The IWorkflowServiceClient Interface

The `IWorkflowServiceClient` interface provides methods, summarized in Table 33-1, for obtaining handles to the various workflow services interfaces.

Table 33-1 IWorkflowServiceClient Methods

Method	Interface
`getTaskService`	`oracle.bpel.services.workflow.task.ITaskService`

Table 33-1 (Cont.) IWorkflowServiceClient Methods

Method	Interface
getTaskQueryService	oracle.bpel.services.workflow.query.ITaskQueryService
getTaskReportService	oracle.bpel.services.workflow.report.ITaskReportService
getTaskMetadataService	oracle.bpel.services.workflow.metadata.ITaskMetadataService
getUserMetadataService	oracle.bpel.services.workflow.user.IUserMetadataService
getRuntimeConfigService	oracle.bpel.services.workflow.runtimeconfig.IRuntimeConfigService
getTaskEvidenceService	oracle.bpel.services.workflow.metadata.ITaskMetadataService

Class Paths for Clients Using SOAP

SOAP clients must have the following JAR files in their class path:

```
$SOA_HOME/soa/modules/oracle.bpm.client_11.1.1/
    oracle.bpm.bpm-services.client.jar
    oracle.bpm.bpm-services.interface.jar
    oracle.bpm.client.jar
    oracle.bpm.web-resources.jar

$SOA_HOME/soa/modules/oracle.bpm.project_11.1.1/
    oracle.bpm.project.catalog.jar
    oracle.bpm.project.draw.jar
    oracle.bpm.project.jar
    oracle.bpm.project.model.jar

$SOA_HOME/soa/modules/oracle.bpm.runtime_11.1.1/
    oracle.bpm.bpm-services.implementation.jar
    oracle.bpm.bpm-services.internal.jar
    oracle.bpm.core.jar
    oracle.bpm.lib.jar
    oracle.bpm.metadata.jar
    oracle.bpm.metadata-interface.jar
    oracle.bpm.papi.jar
    oracle.bpm.xml.jar

$SOA_HOME/soa/modules/oracle.soa.fabric_11.1.1/
    fabric-runtime.jar
    bpm-infra.jar

$SOA_HOME/soa/modules/oracle.soa.workflow_11.1.1/
    bpm-services.jar
    bpm-workflow-datacontrol.jar

$SOA_HOME/soa/modules/
```

```
    soa-startup.jar

$MW_HOME/oracle_common/modules/oracle.webservices_11.1.1/
    wsclient.jar

$MW_HOME/oracle_common/modules/oracle.jrf_11.1.1/
    jrf-api.jar

$MW_HOME/wlserver_10.3/server/lib/
    wlthint3client.jar

${bea.home}/wlserver/server/lib/
    wlfullclient.jar

$ORACLE_HOME/soa/plugins/jdeveloper/external/
    oracle.external.soa.jrf-wsclient-extended.jar

${bea.home}/oracle_common/module/clients/
    com.oracle.webservices.wls.jaxws-owsm-client_12.1.3.jar
```

You can generate the `wlfullclient.jar` file using the commands shown in the code sample below:

```
cd ${bea.home}/wlserver/server/lib
java -jar ../../../modules/com.bea.core.jarbuilder_2.2.0.0.jar
```

Note:

Client applications no longer use the `system\services\config` or `system\services\schema` directories in the class path.

Class Paths for Clients Using Remote EJBs

Clients using remote EJBs must have the following JAR files in their class path:

- `wlfullclient.jar`
- `oracle.external.soa.jrf-wsclient-extended.jar`
- `wlclient.jar`
- `xmlparserv2.jar`
- `xml.jar`
- `bpm-infra.jar`
- `bpm-services.jar`
- `fabric-runtime.jar`

Note:

Client applications no longer use the `system\services\config` or `system\services\schema` directories in the class path.

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.

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.

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.

The code sample below shows how to set 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.

Initiating a Task Programmatically

The code sample below shows how to initiate 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;
```

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 User Metadata Service for more information.

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 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. #unique_1209/unique_1209_Connect_42_CIHDECBA shows sample code from the `login.jsp` file.

```
<%@ 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 Introduction to Building Clients for Workflow Services.

The `statusPage.jsp` file of HelpDeskRequest is used to display the status of help desk requests. The code sample below shows the `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>
```

34

Introduction to Human Workflow Services

This chapter describes for developers how the human workflow services in Oracle SOA Suite 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.

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

SOAP, Enterprise JavaBeans, and Java Support for the Human Workflow Services

Table 34-1 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 34-1 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 criteria 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 34-2 lists the location for the SOAP Web Services Description Language (WSDL) file for each task service.

Table 34-2 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`

Table 34-2 (Cont.) SOAP WSDL Location for the Task Services

Service name	SOAP WSDL location
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`
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 34-3 lists the JDNI names for the different Enterprise JavaBeans.

Table 34-3 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	`TaskQueryService`
User Metadata Service	`UserMetadataService`
Runtime Config Service	`RuntimeConfigService`
Task Report Service	`TaskReportServiceBean`
Task Evidence Service	`TaskEvidenceServiceBean`

For more information about the client library for worklist services, see Building a Custom Worklist Client for details.

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 34-3 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.

   There are different ways to define client properties. For more information, see Configuration Option.

3. Append the JNDI suffix to each Enterprise JavaBeans name shown in Table 34-3 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 the code sample below. You can also use the JAXB `WorkflowServicesClientConfigurationType` object or the `CONNECTION_PROPERTY.EJB_JNDI_SUFFIX` in the `Map<CONNECTION_PROPERTY, String>` properties.

```
<remoteClient>
      <ejbJndiSuffix>_server1</ejbJndiSuffix>
</remoteClient>
```

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, an administrator 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.

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.

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 Obtaining the Workflow Context for a User Previously Authenticated by a JAAS Application.

In the code sample below, the human workflow context is created for user `jcooper`.

```
String adminUser = "...."
String adminPassword = "...."
String realm = "...."

IWorkflowContext adminCtx =
taskQueryService.authenticate(user,password.toCharArray(),realm);

IWorkflowContext behalfOfCtx =
 taskQueryService.authenticateOnBehalfOf(adminCtx,"jcooper");
```

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 the code sample below.

```
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.

Task Service

The task service exposes operations to act on tasks. Table 34-4 describes some of the common operations of the task service. Package `oracle.bpel.services.workflow.task` corresponds to the task service.

For more information about task service, see *Workflow Services Java API Reference for Oracle SOA Suite*.

Table 34-4 Task Service Methods

Method	Description
`acquireTask`	Acquire a task.
`acquireTasks`	Acquire a set of tasks.

Table 34-4 (Cont.) Task Service Methods

Method	Description
`addAttachment`	Add an attachment to a task. **Note:** This API enables a client to call the API to add an attachment before the creation of a task. If the task is not yet created, then the client can call the API with `taskId equals NULL`. However, because the attachment is uploaded before the task is created, Oracle Workflow Services does not enable multiple attachments with the same name to be added to the pre-initiation of a task. If a task is already created, then Oracle Workflow Services keeps only the latest version of the attachment in case multiple attachments have the same name.
`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.
`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.

Table 34-4 (Cont.) Task Service Methods

Method	Description
`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.confi**g > **Server: soa_server1** > **WorkflowConfig** > **human-workflow**. **3.** Click **PushbackAssignee** to view or change the value. **Note:** Pushback is designed to work with single approvers and not with group votes. Pushback from a stage with group vote (or parallel) scenario to another stage is not allowed. Similarly, you cannot push back from a single assignee to a group vote (or parallel) scenario.
`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 34-4 (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.
`updatePriority`	Update the priority of the task and its subtasks for the given task ID. If `UpdatePriorityType` is `INCREMENT` then the task is updated by incrementing the given priority by 1—that is, the priority of the task is raised. If the `UpdatePriorityType` is `DECREMENT`, then the task is updated by decrementing the priority by 1—that is, the priority of the task is lowered, otherwise the task is updated with the given priority.
`updatePriorityOfTasks`	For bulk update of tasks. A list of tasks for which the priority must be updated can be passed as a parameter to this API. The priorities of the list of tasks is updated. It updates the priority of the task and its subtasks.

Table 34-4 (Cont.) Task Service Methods

Method	Description
`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:

- Task Instance Attributes
- *Workflow Services Java API Reference for Oracle SOA Suite*
- Sample workflow-118-JavaSamples, which demonstrates some APIs

Task Query Service

The task query service queries tasks based on a variety of search criteria such as keyword, category, status, business process, attribute values, historical information of a task, and so on. Table 34-5 describes some of the common operations of the task query service. Package `oracle.bpel.services.workflow.query` corresponds to the task query service.

For more information about task query service, see *Workflow Services Java API Reference for Oracle SOA Suite*.

Table 34-5 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.

Table 34-5 (Cont.) Task Query Service Methods

Method	Description
`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 34-5 (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, excluding any tasks that have been claimed by other users. `MY_AND_GROUP_ALL`: 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, including any tasks that have been claimed by other users. `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.

Table 34-5 (Cont.) Task Query Service Methods

Method	Description
`queryViewTasks`	Returns a list of tasks according to the criteria in the specified view. The entire list or paged list of tasks can be returned. Clients can specify additional filter and ordering criteria to those in the view.

For more information, see the following:

- Task Instance Attributes
- *Workflow Services Java API Reference for Oracle SOA Suite* in the documentation library
- Sample workflow-118-JavaSamples, which demonstrates some APIs

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 *Securing Applications with Oracle Platform Security Services*. 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.

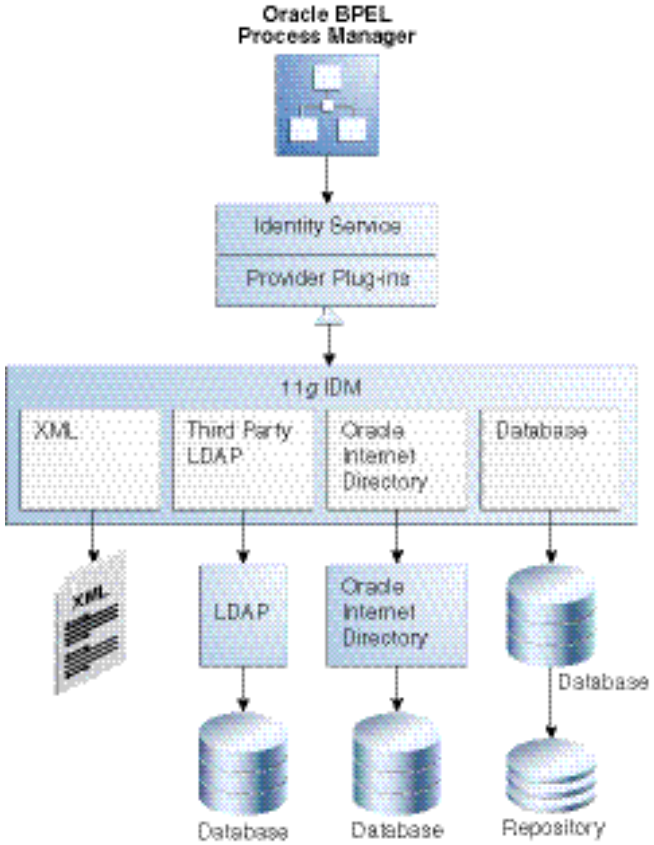
For more information, see the following:

- *Securing Applications with Oracle Platform Security Services* for details about OPSS
- for details about Oracle Identity Management
- for details about OVD

Identity Service Providers

Oracle Identity Management is the only supported provider for release 11*g*, as shown in Figure 34-1.

Figure 34-1 Identity Service Providers



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/technetwork/middleware/id-mgmt/overview/index.html`

Task Metadata Service

The task metadata service exposes operations to retrieve metadata information related to a task. Table 34-6 describes some of the common operations of task metadata service. Package `oracle.bpel.services.workflow.metadata` corresponds to the task metadata service.

For more information about task metadata service, see *Workflow Services Java API Reference for Oracle SOA Suite*.

Table 34-6 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.

Table 34-6 (Cont.) Task Metadata Service Methods

Method	Description
`getOutcomes`	Gets the permitted outcomes of a task. The outcomes are returned with their display values.
`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.
`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 *Workflow Services Java API Reference for Oracle SOA Suite*.

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 34-7 describes some of the common operations of the user metadata service. Package `oracle.bpel.services.workflow.user` corresponds to the user metadata service.

For more information about user metadata service, see *Workflow Services Java API Reference for Oracle SOA Suite*.

Table 34-7 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.

Table 34-7 (Cont.) User Metadata Service Methods

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

For more information, see the following:

- Using for details about the rule configuration and user preference pages
- Sample workflow-118-JavaSamples, which demonstrates some APIs
- *Workflow Services Java API Reference for Oracle SOA Suite*

Task Report Service

The task report service executes a report and receives the results. Table 34-8 shows the list of reports. Package `oracle.bpel.services.workflow.report` corresponds to the task report service. The standard reports shown in Table 34-8 are available as part of installation.

Table 34-8 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.

Table 34-8 (Cont.) Task Report Service

Report	Description
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.

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:

Payload fields that are simple XML types can be mapped directly, or an xpath expression can be specified to select a simple XML type value from a complex payload field.

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 34-9 describes some of the common operations of the runtime config service. Package `oracle.bpel.services.workflow.runtimeconfig` corresponds to the runtime config service.

For more information about runtime config service, see *Workflow Services Java API Reference for Oracle SOA Suite*.

Table 34-9 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.

Table 34-9 (Cont.) Runtime Config Service

Method	Description
`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.
`getDynamicAssignmentFunctions`	Returns a list of dynamic assignment functions 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.
`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:

- Dynamic Assignment and Task Escalation Patterns for additional details
- Using for details about mapped attribute mappings
- Sample workflow-118-JavaSamples, which demonstrates some APIs.
- *Workflow Services Java API Reference for Oracle SOA Suite*

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 Oracle SOA Suite samples for details:

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
```

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.

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

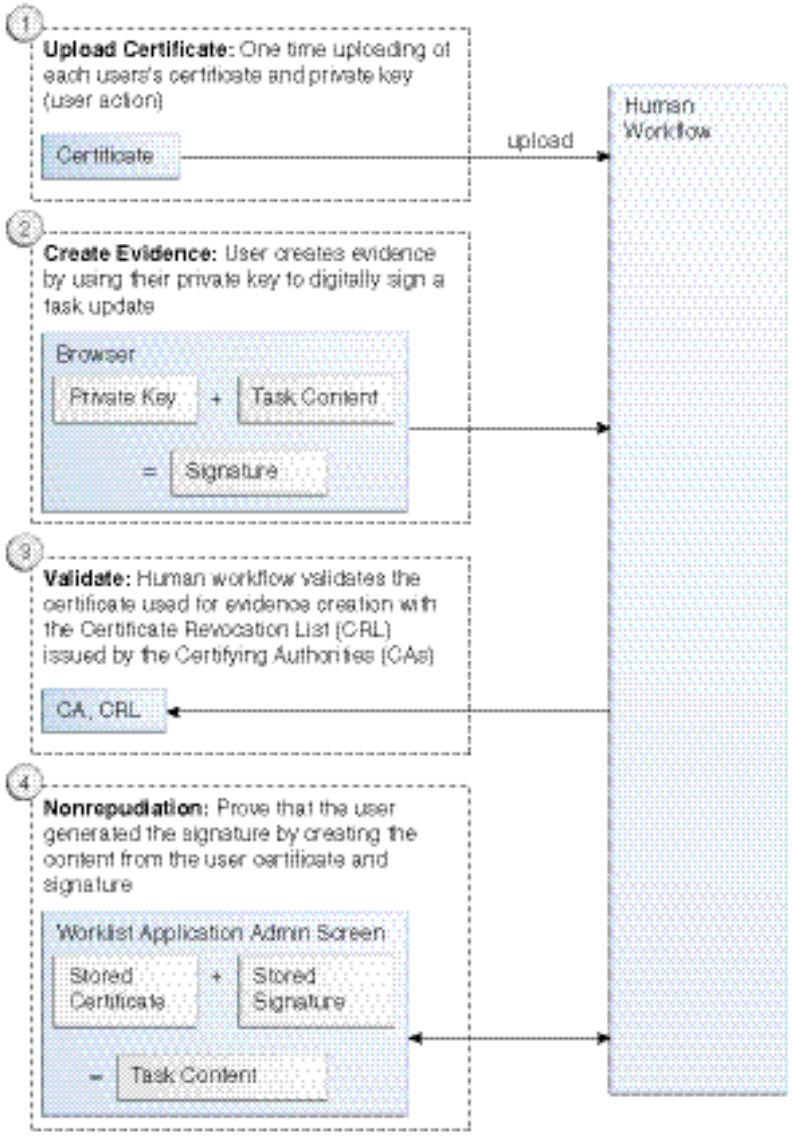
A cryptographically-based digital signature is created when a public key algorithm signs a sender's message with a sender's private key.

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 34-2 provides an example.

Figure 34-2 Digital Signature and Certificate



Note:

- 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

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 34-10.

     Table 34-10 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**.

Interfaces and Methods

Table 34-11 through Table 34-14 describe the methods in the evidence store service. Package `oracle.bpel.services.security.evidence` corresponds to the evidence service.

Table 34-11 ITaskEvidenceService Interface

Method	Description
`createEvidence`	Creates evidence and stores it in the repository for nonrepudiation.
`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 34-12 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.

Table 34-12 (Cont.) Evidence Interface

Method	Description
`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.
`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 34-13 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 34-14 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:

- How to Specify a Workflow Digital Signature Policy for details about specifying digital signatures and digital certificates in the Human Task Editor
- Designing Task Forms for Human Tasks for details about digitally signing a task action in the Oracle BPM Worklist

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 34-15 describes the task attributes that are commonly used and interpreted by applications.

Table 34-15 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.
`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.

Table 34-15 (Cont.) Task Attributes

Task Attribute Name	Description
`task/title`	The title of the task.

Table 34-16 lists the attributes that capture process metadata information.

Table 34-16 Attributes Capturing Process Metadata Information

Attribute	Description
`task/sca/applicationName`	The partition to which the task component that defines this task instance 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.
`task/sca/compositeVersion`	The version of the composite that contains the task component that defines this task instance.
`task/sca/compositeCreatedTime`	The date and time on which the composite flow to which this task instance belongs was started.
`task/sca/flowId`	A unique identifier for the composite flow to which this task instance belongs.

Table 34-17 lists the attachment-related attributes.

Table 34-17 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 34-18 lists the comment-related attributes.

Table 34-18 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. This is set by services when saving comments. If set by client when saving the comment, it is ignored.
`task/userComment/displayNameLanguage`	Set by services when reading comments. This indicates the language in which the `updatedBy displayName` is populated.
`task/userComment/acl`	Not used.
`task/userComment/doesBelongToParent`	If the comment is inherited from parent (example process comment).
`task/userComment/isSystemComment`	Set by services if the comment is set by the workflow system (example, a comment is created if the task goes into alerted state).
`task/userComment/taskId`	The taskId in which the comment was created. For example, if the scope is "BPM", the comment may be visible in a task different than the one in which it was created. Also, for parallel task, the current taskId and comment taskId may be different. This is set by services.
`task/userComment/commentScope`	The values - null, empty or "TASK" implies that the comment is for that task only. The value "BPM" implies that it is for the whole process. The value has to be set to "BPM" when adding comment if you want the comment to be applicable to the whole process.
`task/userComment/updatedBy/id`	ID of the user who updated the comment.
`task/userComment/updatedBy/displayName`	Display name of the user who updated the comment.
`task/userComment/updatedBy/type`	Type of User, Group, or Role of the user who updated the comment.

Table 34-19 lists the attributes manipulated by the workflow services system.

Table 34-19 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.

Table 34-19 (Cont.) Attributes Manipulated by the Workflow Services System

Attribute	Description
`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.
`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.

Table 34-19 (Cont.) Attributes Manipulated by the Workflow Services System

Attribute	Description
`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/subTaskGroupInstanceId`	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.

Table 34-19 (Cont.) Attributes Manipulated by the Workflow Services System

Attribute	Description
`task/systemAttributes/taskNumber`	An integer number that uniquely identifies this task instance.
`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 34-20 lists the mapped attributes.

Table 34-20 Mapped Attributes

Attribute	Description
`task/systemMessageAttributes/*`	The mapped attributes.

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:

- Using the Notification Service
- *Developing Applications with Oracle User Messaging Service*
- *Administering Oracle User Messaging Service* for instructions on configuring notification service delivery channels in Oracle Enterprise Manager Fusion Middleware Control

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 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

The following example demonstrates the 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
```

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 *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

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:

- Using the Notification Service
- *Administering Oracle SOA Suite and Oracle Business Process Management Suite*

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 *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

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 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 How to Notify Recipients of Changes to Task Status.

Notifications can be sent to users involved in the task in various capacities. These users are described in How to Notify 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:

- Using the Notification Service
- Specifying Participant Notification Preferences to configure task notifications in the Human Task Editor
- *Administering Oracle SOA Suite and Oracle Business Process Management 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 *Administering Oracle SOA Suite and Oracle Business Process Management 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.

How to Configure Notification Messages in Different Languages

A notification consists of four types of data generated from multiples sources and internationalized differently.

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()
```

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

For more information, see How to Specify Multilingual Settings.

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.

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**.

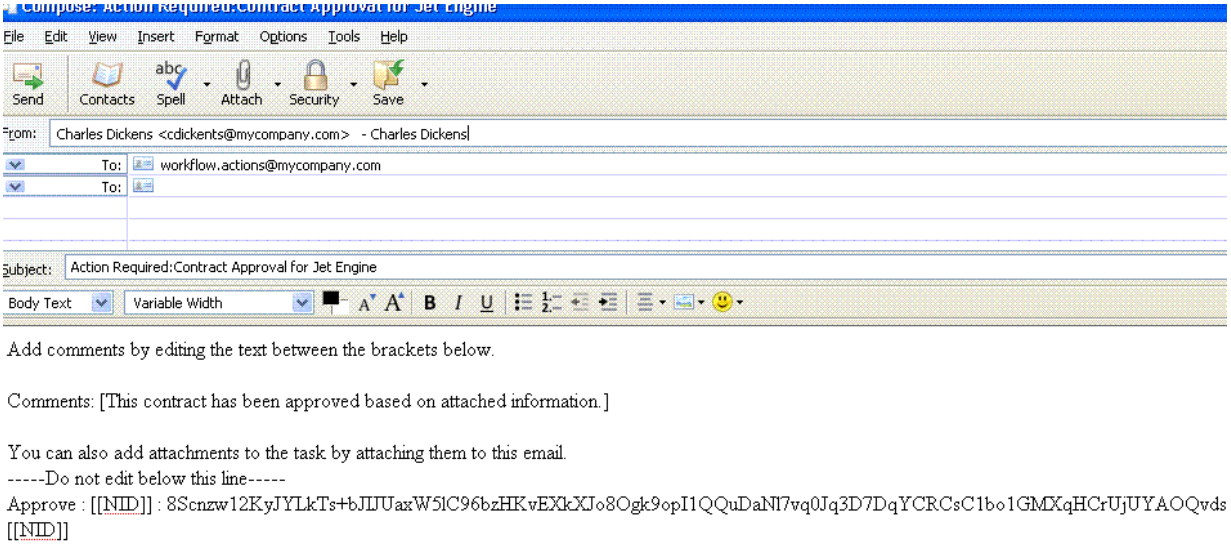
   When an actionable email arrives, perform the following tasks.

3. Set properties such as incoming server, outgoing mail server, outgoing user name and password, and others from the Oracle User Messaging Service section of Oracle Enterprise Manager Fusion Middleware Control.

4. In the Workflow Notification Properties page of Oracle Enterprise Manager Fusion Middleware Control, set the notification mode to **ALL** or **EMAIL**.

5. Click the **Approve** link to invoke a new email window with approval data. Figure 34-3 provides details.

Figure 34-3 Actionable Notifications

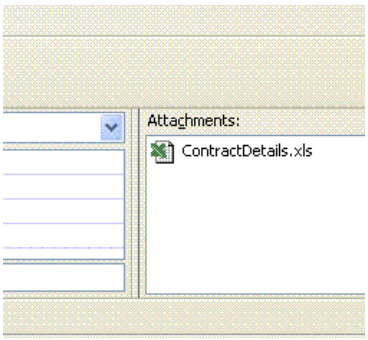


6. Add comments in the comments section of the approval mail. For example:

   ```
   This contract has been approved based on the attached information.
   ```

7. Add attachments as needed, as shown in Figure 34-4.

Figure 34-4 Attachment to an Actionable Email



8. 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.

9. Click **Send**.

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 *Administering Oracle User Messaging Service* and *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

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 How to Make Email Messages Actionable.

How to Send Inbound Comments

To send inbound comments:

1. Add comments in the actionable email reply between `Comments[[' and ']]`, as shown in Figure 34-3. Those contents are added as task comments. For example, `Comments[[looks good]]`.

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 How to Secure Notifications to Exclude Details.

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 .

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:

   - Specify the number of times reminders are sent. The values are `No Reminders`, `Remind Once`, `Remind Twice`, `Remind Three Times`.
   - Specify when the reminder must be sent. Select the values from `Day`, `Hour`, `Minutes`, and select `Before Expiration` or `After Expiration`. The values `Before Expiration` or `After Expiration` are related to the expiration of the task.

   For more information, see How to Set Up Reminders.

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. The code sample below shows the `WorkflowLabels.properties` file. For more information, see Global Resource Bundle – 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.
```

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`.

   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:

   - How to Send Task Attachments with Email Notifications
   - *Developing Applications with Oracle User Messaging Service*

Assignment Service Configuration

This section describes how to configure the assignment service with dynamic assignment functions. It contains these topics:

- Dynamic Assignment and Task Escalation Patterns
- Dynamically Assigning Task Participants with the Assignment Service
- Custom Escalation Function

Dynamic Assignment and Task Escalation Patterns

When tasks are assigned to a group, application role, or list of users a single user must claim a task to act on it. However, you can also automatically send work to users by using various dispatching mechanisms.

Automatic task dispatching is done through dynamic assignment patterns. Dynamic assignment patterns select a particular user or group from either a group or a list of users or groups. Similarly, when a task is escalated, a task escalation pattern can be used to determine the user to whom the task should be escalated to. Several patterns are provided out of the box. However, you can also create your own patterns for dynamic assignment and task escalation and register them with the workflow service. Table 34-21 describes the three dynamic assignment patterns and one task escalation pattern that are provided out-of-the-box.

Table 34-21 Dynamic Assignment Patterns

Assignment Pattern	Type	Description
`LEAST_BUSY`	Dynamic assignment	Picks the user or group with the least number of tasks currently assigned to it.
`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.
`MANAGERS_MANAGER`	Task escalation	Picks the manager's manager.

These patterns all check a user's vacation status. A user that is currently unavailable is not automatically assigned tasks.

Dynamic assignment patterns can be used when defining a task participant, as described in How to Configure the Single Participant Type. They can also be used with task-assignment rules allowing end-users to specify dynamic assignment of tasks to the members of groups that they manage, as described in How To Create Group Rules.

The dynamic assignment patterns can also be called by using an xpath function in any xpath expression in the task definition.

The signature of the function is:

```
hwf:dynamicTaskAssign(patternName, participants, inputParticipantType,
targetAssigneeType, isGlobal, invocationContext, parameter1, parameter2, ...,
parameterN)
```

The parameters are:

- `patternName`: Mandatory. Name of the pattern to use
- `participants`: Mandatory. The participant or participants to select the assignee from. Can be a string or element containing a participant name or a comma-separated list of participant names, or a set of elements containing participant names or comma-separated lists of participant names. Participants must all be of the same type.

- `inputParticipantType`: Mandatory. The type of the input participants (user, group, or application_role)
- `targetAssigneeType`: Mandatory. The type of assignee to select (user, group, or application_role). Value must match the context in which the function is being used (for example, must be user if dynamically selecting an owner user. If the inputParticipantType is user, the only valid value here is user.
- `isGlobal`: Boolean value that indicates if the pattern should be assessed using tasks of all types, or just tasks of the same type as the current task. Optional - defaults to false.
- `invocationContext`: String to uniquely identify where this function is being used. If not specified, a default context is assigned.
- `parameterN`: Some dynamic assignment patterns allow parameters to be specified. The parameter values can be specified as name-value pairs, using an "=" character as a delimiter - for example, "TIME_PERIOD=7"

Example usages:

```
hwf:dynamicTaskAssign("LEAST_BUSY","jcooper,jstein,mtwain","user","user","true","Erro
rAssignee")

hwf:dynamicTaskAssign("MOST_PRODUCTIVE",task:task/task:payload/
task:users,"user","user","false","OwnerUser","TIME_PERIOD=7")

hwf:dynamicTaskAssign("LEAST_BUSY","DeveloperRole","application_role","group"):
```

Before 12c Release 1 (12.1.3), dynamic assignment could be achieved by using the XPath functions `wfDynamicUserAssign` and `wfDynamicGroupAssign`. These XPath functions have been deprecated in 12c Release 1 (12.1.3). They can still be used, but Oracle recommends that you migrate any existing usage of these XPath functions to the new `dynamicTaskAssign` function.

How to Implement a Dynamic Assignment Pattern

Follow these procedures to implement your own dynamic assignment pattern.

To implement dynamic assignment patterns:

Write a Java class that implements the following interface:

```
oracle.bpel.services.workflow.assignment.dynamic.IDynamicAssignmentPattern
```

Implementations must provide support for selecting a single assignee from a list of participants (all of the same type) by implementing the method `getAssigneeFromParticipantList`.

An implementation does not have to support all assignee types. The interface provides the method `getSupportedAssigneeType` to enable the implementation to specify which types of assignee it supports.

Implementations can accept input parameters to specify selection criteria, the Dynamic Assignment Framework validates these input parameters, and the implementation can define its parameters (if any) in the method `getPatternParameters()`.

An implementation can also accept initialization parameters, which are set when the implementation is initialized by the framework. The parameter values are defined in the human workflow configuration (either using `configMBean`, or by Human

Workflow Service Engine configuration in Oracle Enterprise Manager Fusion Middleware Control), where the dynamic assignment pattern is registered.

For convenience, the framework provides the class `AbstractDynamicAssignmentPattern` which implements some common functionality. Assignment pattern implementations can extend this abstract class, to save implementing some parameter and localization support.

Before 11g (11.1.1.6.0), custom dynamic assignment patterns were implemented using one or both of the following interfaces:

```
oracle.bpel.services.workflow.assignment.dynamic.IDynamicGroupAssignmentFunction
oracle.bpel.services.workflow.assignment.dynamic.IDynamicUserAssignmentFunction
```

These interfaces do not offer all the features available in the `IDynamicAssingmentPattern` interface, and have been deprecated. The Dynamic Assignment Framework remains backward compatible with implementations that use the old interface, but Oracle recommends that you migrate any implementations you have to use the new interface.

For information about the Javadoc for dynamic assignment interfaces and utilities, see *Oracle Fusion Middleware Workflow Services Java API Reference for Oracle BPEL Process Manager*.

How to Configure Dynamic Assignment Patterns

Dynamic assignment patterns are configured along with other human workflow configuration parameters in Oracle Enterprise Manager Fusion Middleware Control.

Each dynamic assignment has two mandatory parameters:

- `name`:

  The name of the pattern

- `classpath`:

  The fully qualified class name of the class that implements the pattern.

In addition, each pattern can optionally have any number of properties. These properties are simple name-value pairs that are passed as initialization parameters to the pattern.

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 patterns.

Two of the out-of-the-box patterns 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 pattern can maintain `ROUND_ROBIN` counts. The dynamic assignment pattern 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` pattern.

- `MOST_PRODUCTIVE`

  The parameter `DEAFULT_TIME_PERIOD` specifies 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 pattern.

For more information about configuring the dynamic assignment functions from Oracle Enterprise Manager Fusion Middleware Control, see *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

How to Configure Display Names for Dynamic Assignment Patterns

The runtime config service provides methods for returning a list of available user and group dynamic assignment patterns. These patterns return both the name of the pattern, and a user-displayable label and description for the pattern and its parameters. The patterns support localization of the display name, so that it displays in the appropriate language for the context user. These patterns are used by Oracle BPM Worklist and the JDeveloper Human Task Editor to show a list of available dynamic assignment patterns.

The dynamic assignment framework provides methods allowing pattern implementations to provide human-readable display names and descriptions for patterns and their parameters.

The out-of-the-box pattern implementations, and custom implementations that extend the `AbstractDynamicPattern` class use the `WorkflowLabels.properties` resource bundle file to configure these display strings.

To configure display names for dynamic assignment patterns:

Specify display names and descriptions (and appropriate translations) for your dynamic assignment patterns and their parameters by adding entries to the resource property file `WorkflowLabels.properties`, and associated resource property files in other languages. This file should be placed in the class path identified in the workflow configuration parameter `workflowCustomizationsClasspathURL`, at the path

```
oracle/bpel/services/workflow/resource/WorkflowLabels.properties
```

Entries for dynamic assignment patterns must be of the following form:

```
DYN_ASSIGN_FN.[pattern name]=Pattern Display Name

DYN_ASSIGN_DESCR.[pattern name]=Function Description

DYN_ASSIGN_PARAM_LABEL.[pattern name].[parameter name]=Parameter Display Name

DYN_ASSIGN_PARAM_LABEL.[pattern name].[parameter name]=Parameter Description
```

For instance, the entries for the `MOST_PRODUCTIVE` pattern are:

```
DYN_ASSIGN_FN.MOST_PRODUCTIVE = Most Productive

DYN_ASSIGN_DESCR.MOST_PRODUCTIVE = Picks the user, group or application role that
has completed the highest number of tasks within a certain time period. For group
and application roles the total number of tasks completed by all the users who
are direct members of that group or role are counted. The time period to use can
be specified using the Time Period parameter. If no time period is specified,
then the default value specified in the dynamic assignment configuration for the
instance is used.

DYN_ASSIGN_PARAM_LABEL.MOST_PRODUCTIVE.TIME_PERIOD = Time Period

DYN_ASSIGN_PARAM_DESCR.MOST_PRODUCTIVE.TIME_PERIOD = The previous number of days
over which to count the number of completed tasks. If not specified, the default
value defined in the human workflow dynamic assignment configuration is used.
```

Adding entries to these files for dynamic assignment patterns 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 with the Oracle SOA Suite samples.

How to Implement a Task Escalation Pattern

Task escalation functions are very similar to dynamic assignment patterns, but perform a different function (determining to whom a task is assigned when it is escalated). Custom implementations must implement a different interface (`IDynamicTaskEscalationPattern`).

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.

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).

Example of Assignment Service Implementation

Note:

- The assignment service class cannot be stateful. This is 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.

The code sample below provides a sample `IAssignmentService` implementation named `TestAssignmentService.java`.

```
/* $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;
    }

}
```

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 Oracle SOA Suite samples.

   Note:

   - You cannot create different versions of the assignment service for use in different BPEL processes unless you change package names or class names.
   - Java classes and JAR files in the suitcase are not available in the class path and therefore cannot be used as a deployment model for the assignment service.
   - The steps must be repeated for each node in a cluster.

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 How to Specify Escalation Rules.

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.

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.

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 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. 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 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.

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.

To customize the file:

1. Edit the properties file.
2. Add the customized class to the class-path used for workflow services. Ensure that customized file is located before the default class in the class-path.
3. Save the customized file to the following directory:

   directory_path`/oracle/bpel/services/workflow/resource/WorkflowLabels.properties`

4. 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, or to store them in the MDS repository for the SOA server, and use the '`oramds`' URL protocol, 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 Oracle SOA Suite samples.

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.

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.

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`*

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.

Introduction to Human Workflow Client Integration with Oracle WebLogic Server Services

This section describes how human workflow clients integrate with Oracle WebLogic Server services.

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. The first code sample in Workflow Client Configuration File - wf_client_config.xml provides details.

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`.

The second code sample in Workflow Client Configuration File - wf_client_config.xml provides details.

In the second example, `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, you can use the factory method from the `WorkflowServiceClientFactory` class. See the code sample below:

```
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 the code sample below can be used:

```
public static IWorkflowServiceClient getWorkflowServiceClient(Map<CONNECTION_
PROPERTY, String> properties, String serverName,    Logger logger) throws
 WorkflowException
```

Task Query Service Client Code

The code sample below provides an example of the 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.example.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);
   }

}
```

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.

JAXB Object

You can use the JAXB object to define the client configuration. The code sample below shows how to use the `WorkflowServiceClientFactory` method.

```
public static IWorkflowServiceClient getWorkflowServiceClient(String
clientType,WorkflowServicesClientConfigurationType wscc,Logger logger) throws
WorkflowException
```

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 the code sample below. The configuration defines a server named `default`. The XML file must go into the client application's EAR file.

```
<workflowServicesClientConfiguration>
server name="default" default="true">

<remoteClient>
   <serverURL>t3://myhost.us.example.com:7001</serverURL>
   <userName>weblogic</userName>
   <password>weblogic</password>
   <initialContextFactory>weblogic.jndi.WLInitialContextFactory
      </initialContextFactory>
   <participateInClientTransaction>false</participateInClientTransaction>
</remoteClient>

<soapClient>
   <rootEndPointURL>http://myhost.us.example.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 following code sample shows an example of a 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.example.com:7001</serverURL>
         <initialContextFactory>weblogic.jndi.WLInitialContextFactory</
initialContextFactory>
         <participateInClientTransaction>false</participateInClientTransaction>
      </remoteClient> -->
      <soapClient>
         <rootEndPointURL>http://myhost1.us.example.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.example.com:7001</serverURL>
         <initialContextFactory>weblogic.jndi.WLInitialContextFactory</
initialContextFactory>
         <participateInClientTransaction>false</participateInClientTransaction>
      </remoteClient> -->
      <soapClient>
```

```
        <rootEndPointURL>http://myhost2us.example.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.

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 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. The code sample below provides details.

```
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. The code sample below provides details.

```
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 shown below can be used:

```
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, as shown below:

```
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.

The code sample below provides an 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.example.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);
```

The code sample below provides an example for a 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);
```

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. The code sample below shows how to pass a logger to the workflow service clients:

```
java.util.logging.Logger logger = ....;

IWorkflowServiceClient client =
WorkflowServiceClientFactory.getWorkflowServiceClient(WorkflowServiceClientFactory
.REMOTE_CLIENT, properties, logger);
```

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 the code sample below:

```
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.

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.

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.

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 Workflow Client Configuration in the Property Map.

- `IWorkflowServiceClientConstants.CONNECTION_PROPERTY.EJB_SECURITY_PRINCIPAL`
- `IWorkflowServiceClientConstants.CONNECTION_PROPERTY.EJB_SECURITY_CREDENTIALS`

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 *Securing Applications with Oracle Platform Security Services*.

For information about interacting with custom identity stores, visit the following URL:

`http://www.oracle.com/technetwork/middleware/id-mgmt/overview/index.html`

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.

Client configuration

To enable identity propagation, the client configuration must specify a special propagation mode.

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 the code sample below is optional. If the identity propagation mode is set, you run by default in dynamic mode.

  ```
  properties.put(IWorkflowServiceClientConstants.CONNECTION_PROPERTY.SOAP_
  IDENTITY_PROPAGATION_MODE , "dynamic");
  ```

By default, SAML-token constructs dynamic policy based on the following security policy URI: `oracle/wss10_saml_token_client_policy`. Logging is not used. To overwrite the default policy URI, the client can add the code shown below:

```
properties.put(CONNECTION_PROPERTY.SECURITY_POLICY_URI     "oracle/wss10_saml_
token_client_policy");
properties.put(CONNECTION_PROPERTY.MANAGEMENT_POLICY_URI , "oracle/log_policy");
```

The code sample below shows the SAML 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.example.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.

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 below:

```
<soapClient>
        <rootEndPointURL>http://myhost.us.example.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 *Administering Web Services*.

Identity Propagation Mode Setting Through the JAXB Object

You can programmatically set the identity propagation mode with the JAXB object.

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. The code sample below provides details.

```
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.example.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. The code sample below provides details.

```
<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 *Administering Web Services*.

Client JAR Files

A client application without identity propagation must have the `bpm-services.jar` file in its class path. For 12c Release 1 (12.1.3), the client class path requires the files shown below:

```
$fmwhome/wlserver/server/lib/wlfullclient.jar
$fmwhome/wlserver/lib/weblogic.jar
$fmwhome/wlserver/server/lib/wlclient.jar
$fmwhome/oracle_common/modules/clients/com.oracle.webservices.fmw.client_
12.1.3.jar
$fmwhome/soa/soa/modules/com.oracle.webservices.fmw.client_12.1.3.jar
$fmwhome/oracle_common/modules/oracle.xdk_12.1.3/xml.jar
$fmwhome/oracle_common/modules/oracle.nlsrtl_11.2.0/orai18n-mapping.jar
$fmwhome/soa/soa/modules/oracle.soa.fabric_11.1.1/bpm-infra.jar
$fmwhome/soa/soa/modules/oracle.soa.workflow_11.1.1/bpm-services.jar
$fmwhome/soa/soa/modules/soa-startup.jar
```

The `wlfullclient.jar` file must be generated.

- Generate the `wlfullclient.jar` as follows:

  ```
  cd $fmwhome/wlserver/server/lib
  java -jar ../../modules/com.bea.core.jarbuilder_2.2.0.0.jar
  ```

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 *Workflow Services Java API Reference for Oracle SOA Suite*.

Database Views for Oracle Workflow

This section describes database views that enable queries against the Oracle workflow services schema to receive reports. Table 34-22 lists the reports exposed in Oracle BPM Worklist and the database views corresponding to these reports.

Table 34-22 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`

Unattended Tasks Report View

Table 34-23 describes the `WFUNATTENDEDTASKS_VIEW` report view.

Table 34-23 Unattended Tasks Report View

Name	Type
`TASKID`[1]	`VARCHAR2(64)`
`TASKNAME`	`VARCHAR2(200)`
`TASKNUMBER`	`NUMBER`
`CREATEDDATE`	`DATE`
`EXPIRATIONDATE`	`DATE`
`STATE`	`VARCHAR2(100)`
`PRIORITY`	`NUMBER`
`ASSIGNEEGROUPS`	`VARCHAR2(2000)`

1 NOT NULL column

For example:

- Query unattended tasks that have an expiration date of next week, as shown below:

```
SELECT tasknumber, taskname, assigneegroups FROM WFUNATTENDEDTASKS_VIEW
 WHERE expirationdate > current_date AND expirationdate < current_date +
 7;
```

- Query unattended tasks for `mygroup`, as shown below:

```
SELECT tasknumber, taskname, assigneegroups FROM WFUNATTENDEDTASKS_VIEW
 WHERE 'mygroup' IN assigneegroups;
```

- Query unattended tasks created in the last 30 days, as shown below:

```
SELECT tasknumber, taskname, assigneegroups FROM WFUNATTENDEDTASKS_VIEW
 WHERE createddate > current_date -30;
```

Task Cycle Time Report View

Table 34-24 describes the `WFTASKCYCLETIME_VIEW` report view.

Table 34-24 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 below:

```
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 below:

```
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 below:

```
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 below:

```
SELECT avg(cycletime) FROM  WFTASKCYCLETIME_VIEW  WHERE enddate >
  (current_date - 7);
```

- Query tasks that took more than seven days to complete, as shown below:

```
SELECT taskname, avg(cycletime) FROM WFTASKCYCLETIME_VIEW WHERE cycletime
 > ((current_date +7) - current_date) GROUP BY taskname;
```

Task Productivity Report View

Table 34-25 describes the WFPRODUCTIVITY_VIEW report view.

Table 34-25 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 below:

```
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 below:

```
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 below:

```
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 below:

```
SELECT username, taskname, count(taskid) FROM WFPRODUCTIVITY_VIEW WHERE
 lastupdateddate > (current_date -100) GROUP BY username, taskname;
```

Task Priority Report View

Table 34-26 describes the WFTASKPRIORITY_VIEW report view.

Table 34-26 Task Priority Report View

Name	Type
TASKID[1]	VARCHAR2(64)
TASKNAME	VARCHAR2(200)

Table 34-26 (Cont.) Task Priority Report View

Name	Type
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 below:

```
SELECT updatedby, priority, count(taskid) FROM  WFTASKPRIORITY_VIEW GROUP
 BY  updatedby, priority;
```

- Query task-to-outcome distribution, as shown below:

```
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 below:

```
SELECT priority, count(taskid) FROM WFTASKPRIORITY_VIEW WHERE
 updatedby='jstein' GROUP BY priority;
```

Part VI

Using Binding Components

This section describes how to use binding components.

This part contains the following chapters:

35

Getting Started with Binding Components

This chapter describes the supported service and reference binding component types and technologies that you can integrate in a SOA composite application. Supported binding components include web services, HTTP binding, JCA adapters, Cloud adapters, Oracle Business Activity Monitoring (BAM), Oracle B2B, Oracle Healthcare, ADF-BC services, Enterprise JavaBeans (EJB) services, Managed File Transfer (MFT), Representational State Transfer (REST) services, and direct binding services. Creation of tokens for use in the binding URLs of external references is also described.

This chapter includes the following sections:

For more information, see Adding Service Binding Components and Adding Reference Binding Components.

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 REST binding).

- References

  Enable messages to be sent from the SOA composite application to external services in the outside world. For REST bindings, a Web Application Description Language (WADL) file advertises the capabilities to external applications.

Figure 35-1 shows an **OrderBookingComposite** project 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 BAM Server in the outside world.

Figure 35-1 Service and Reference Binding Components

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

- SOAP web services
- HTTP binding
- JCA adapters
- Oracle E-Business Suite
- Oracle BAM 11*g* (This adapter can only connect to an Oracle BAM 11g server.)
- Oracle B2B
- Oracle Healthcare
- Oracle Managed File Transfer (MFT)
- ADF-BC services
- EJB services
- Direct binding services
- REST binding
- Cloud adapters

These technologies are described in the following sections.

SOAP Web Services

This service enables you to integrate applications with a standards-based web service using the Simple Object Access Protocol (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 How to Define the Interface (WSDL) for a Web Service.

For information about adding Message Transmission Optimization Mechanism (MTOM) attachments to web services, see Sending and Receiving MTOM-Optimized Messages to SOA Composite Applications.

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 35-2 shows the support for WS-AT at the bottom of the Create Web Service dialog.

Figure 35-2 WS-AT Support in Create Web Service Dialog

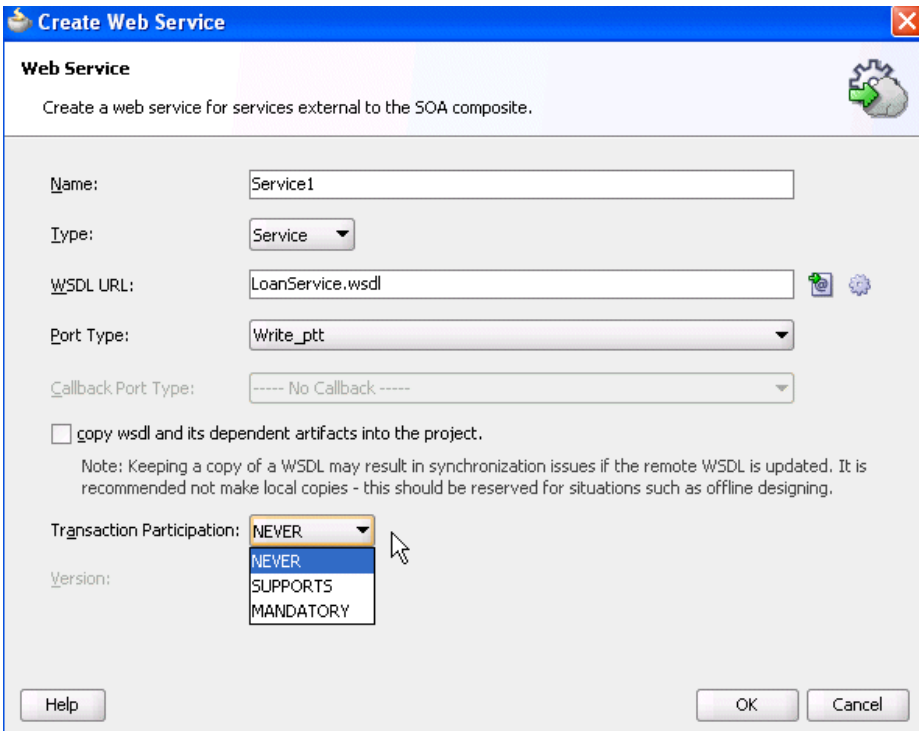


Table 35-1 describes the WS-AT fields. For a description of the remaining fields in the Create Web Service dialog, see How to Define the Interface (WSDL) for a Web Service.

Table 35-1 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.
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 the following example:

```
  <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 *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

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

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

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 How to Add a BPEL Process Service Component, How to Define Deployment Descriptor Properties in the Property Inspector, and Transaction Semantics.

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 *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

HTTP Binding Service

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

You drag the **HTTP** service from the Components window 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.

Supported Interactions

Table 35-2 shows the supported verbs, payloads, and operations for the inbound and outbound directions.

Table 35-2 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
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

Table 35-2 (Cont.) Supported Verbs, Payloads, and Operations

Direction	Verb	Payload Type	Operation	Supported?
Outbound	POST	XML	Request-response	Yes

Table 35-3 shows the supported XSD types for the inbound and outbound directions.

Table 35-3 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`

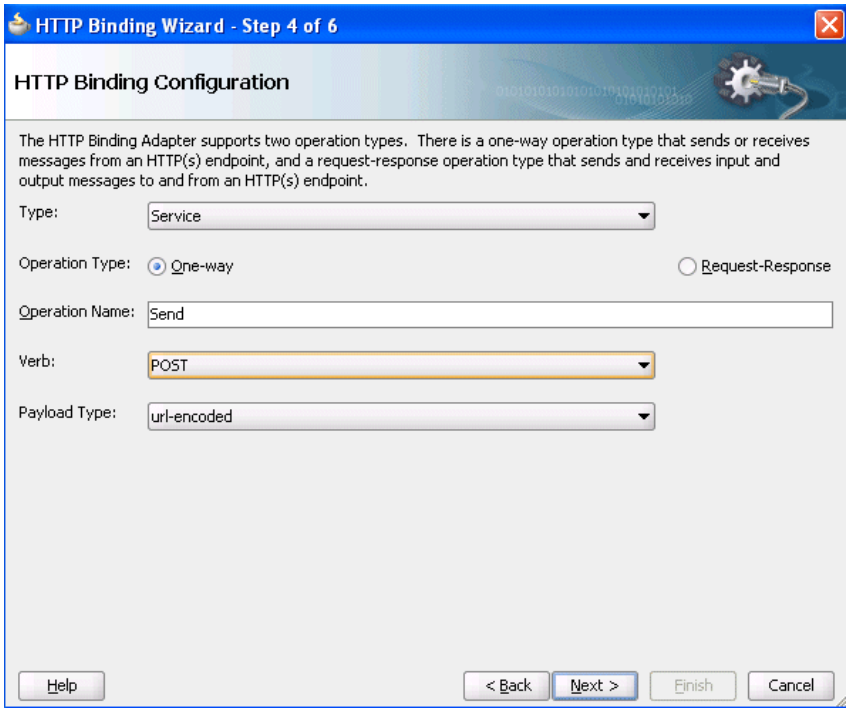
How to Configure the HTTP Binding Service

To configure the HTTP binding service:

1. Invoke the HTTP Binding Wizard to configure HTTP binding by dragging the **HTTP** icon from the Components window.
2. Provide appropriate responses on the Welcome, Service Name, and Adapter Interface pages.

   The HTTP Binding Component page of the wizard enables you to specify the operation type, verb, and payload type. Figure 35-3 provides details.

Figure 35-3 Create HTTP Binding Wizard - HTTP Binding Configuration Page



3. 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 of 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:

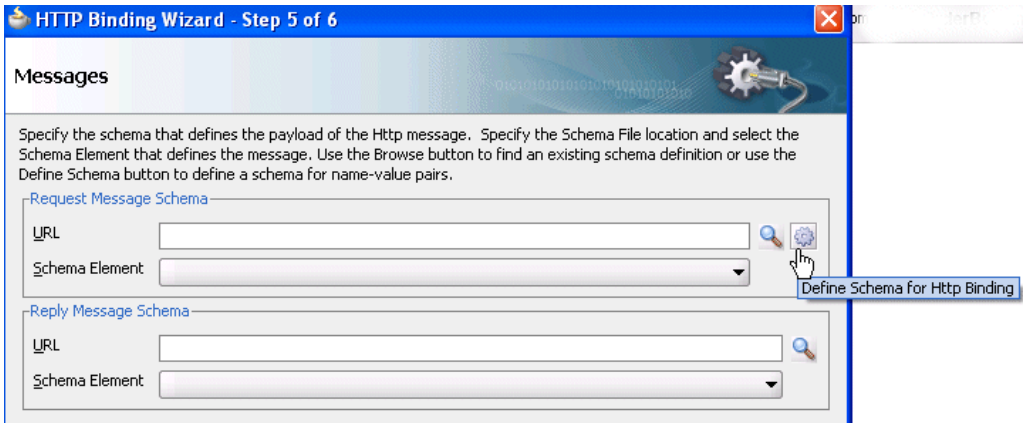
   Secure HTTP (HTTPS) is supported in both the inbound and outbound directions.

   ---

4. Click **OK**.

5. Browse for an existing request message schema or define your own schema with the links to the right of the **URL** field on the Messages page. Figure 35-4 provides details.

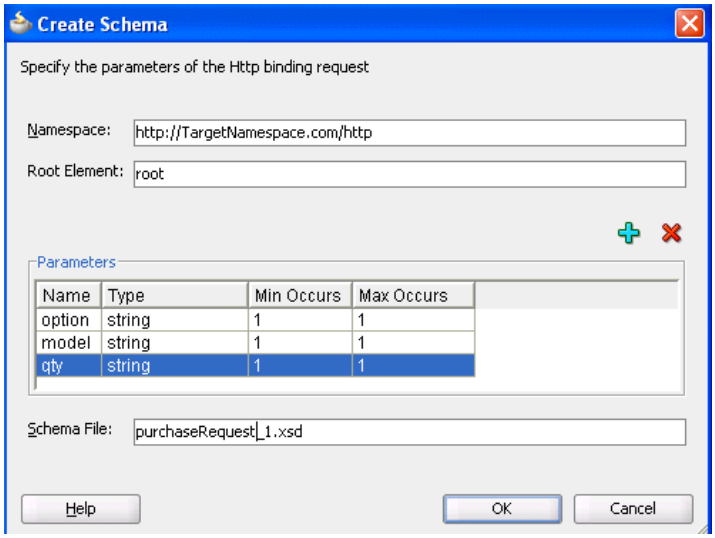
Figure 35-4 Create HTTP Binding Wizard - Messages Page



6. Click **OK**.

7. If you select to define your own schema, you are prompted to specify the element names, data types, minimum occurrence value, and maximum occurrence value in the Create Schema dialog. Figure 35-5 provides details.

Figure 35-5 Create HTTP Binding Wizard - Create Schema Page



8. Click **OK**.

   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.

How to Enable Basic Authentication for HTTP Binding

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. 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**.

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, JDE World, and others)
- Oracle User Messaging Service
- Lightweight Directory Access Protocol (LDAP) server
- Coherence cache

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

Database Adapter

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

For more information, see "Oracle JCA Adapter for Database" of *Understanding Technology Adapters*.

File Adapter

The file adapter enables a BPEL process or an 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 JCA Adapter for Files/FTP" of *Understanding Technology Adapters*.

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 JCA Adapter for Files/FTP" of *Understanding Technology Adapters*.

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 JCA Adapter for AQ" of *Understanding Technology Adapters*.

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 JCA Adapter for JMS" of *Understanding Technology Adapters*.

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. The MQ Series provides a queuing infrastructure that provides guaranteed message delivery, security, and priority-based messaging.

For more information, see "Oracle JCA Adapter for MQ Series" of *Understanding Technology Adapters*.

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 JCA Adapter for Sockets" of *Understanding Technology Adapters*.

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 *Understanding Technology Adapters*.

Oracle User Messaging Service Adapter

The Oracle User Messaging Service supports messaging channels such as email, secure messaging service (SMS), instant messaging, and voice. The Oracle User Messaging Service provides a messaging proxy between the BPEL processes or Oracle Mediator service component and the external world. The Oracle User Messaging Service provides two-way messaging (inbound and outbound).

For more information, see "Oracle JCA Adapter for UMS" of *Understanding Technology Adapters*.

LDAP Adapter

The LDAP adapter defines both asynchronous and synchronous interfaces to send requests to and receive responses from LDAP directory servers. The LDAP adapter enables processes to search, compare, and modify LDAP directories using the LDAP protocol.

For more information, see "Oracle JCA Adapter for LDAP" of *Understanding Technology Adapters*.

Coherence Adapter

A Coherence cache is a collection of data objects that serves as an intermediary between the database and client applications. Database data can be loaded into a cache and made available to different applications. A Coherence cache reduces load on the database and provides faster access to database data. Objects in the cache can be either XML or Plain Old Java Objects (POJOs). The Coherence adapter enables you to perform the following operations against a Coherence cache.

- Add an item
- Obtain an item
- Remove an item
- Query for an item

For more information, see "Oracle JCA Adapter for Coherence" of *Understanding Technology Adapters* and Section "Reading the Shipping Provider from Cache with the Coherence Adapter" of *Understanding Oracle SOA Suite*.

Oracle E-Business Suite 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 E-Business Suite Adapter User's Guide*.

Oracle BAM 11*g* Adapter

The Oracle BAM 11*g* adapter enables you to integrate Java EE applications with an Oracle BAM 11*g* server to send data. This adapter can only connect to an Oracle BAM 11*g* server.

Dragging a **BAM 11*g*** icon into a swimlane of the SOA Composite Editor invokes the Adapter Configuration Wizard for specifying configuration properties.

For more information, see *Monitoring Business Activity with Oracle BAM*.

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 provides for 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 a **B2B** icon into a swimlane of the SOA Composite Editor invokes the B2B Configuration Wizard for specifying configuration properties.

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

Oracle Healthcare Adapter

The Oracle Healthcare adapter enables you to create an end-to-end health care integration process in a SOA composite application. The Healthcare adapter establishes the connection between a SOA composite application and the external health care applications with which data is shared or with an internal topic or queue, where data can be made available internally or to other systems. You can use other Oracle SOA Suite components in your composite application, including BPEL processes, Oracle Mediator components, a variety of adapters, and so on.

The Healthcare Configuration Wizard in Oracle JDeveloper lets you add health care integration binding components to a SOA composite application as follows:

- The component is used as a service (inbound) to receive messages from external systems and deliver them to SOA composite applications. Oracle SOA Suite for health care integration is the entry point to the SOA composite application.
- The component is used as a reference (outbound) to send messages from the SOA composite application to external applications.

As you follow the steps in the Healthcare Configuration Wizard, you are prompted to select a document definition created in Oracle SOA Suite for health care integration.

You can launch Oracle SOA Suite for health care integration from the wizard to create a document definition if the right one does not already exist. This is the payload, or message, that you are receiving from or sending to external systems.

For more information, see *Healthcare Integration User's Guide for Oracle SOA Suite*.

Oracle MFT

Oracle MFT enables you to transfer files to and from many endpoint types, such as the following:

- Embedded FTP or sFTP server
- Remote FTP or sFTP server
- Directories
- SOAP web service endpoints
- Oracle SOA Suite SOAP web service endpoints
- Oracle Service Bus web service endpoints
- Oracle B2B partners and Oracle Healthcare endpoints
- Oracle Data Integrator web service endpoints

For more information, see *Using Oracle Managed File Transfer*.

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

For more information about Oracle ADF, see the following:

- Delegating XML Data Operations to Data Provider Services
- Using Standalone SDO-based Variables
- *Developing Fusion Web Applications with Oracle Application Development Framework*
- *Developing Web User Interfaces with Oracle ADF Faces*

EJB Adapter

The EJB adapter enables Enterprise JavaBeans and SOA composite applications to interact by passing Java interfaces (does not use a WSDL file to define the interface) or SDO parameters (uses 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** icon into a swimlane of the SOA Composite Editor invokes the Create EJB Service dialog for specifying configuration properties.

For more information, see Integrating Enterprise JavaBeans with Composite Applications .

Direct Binding Adapter

The direct binding adapter 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** icon into a swimlane of the SOA Composite Editor invokes the Create Direct Binding dialog for specifying configuration properties.

For more information about direct binding, see Using Direct Binding to Invoke Composite Services .

For information about the Direct Binding Invocation API, see *Java API Reference for Infrastructure Management*.

For more information about OSB, see *Developing Services with Oracle Service Bus*.

REST Binding

REST is an architecture for designing network applications. RESTful applications use HTTP requests to post data (create and update), get data (for example, make queries), and delete data. REST provides an alternative to using web services. A SOA composite can be REST-enabled or invoke an existing REST service through the REST adapter.

For more information, see Integrating REST Operations in SOA Composite Applications.

Cloud Adapters

The cloud adapters, such as the SalesForce adapter, enable you to send and receive messages from a cloud server.

For more information, see *Understanding Technology Adapters*.

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 Components window.

How to Integrate a Binding Component in a SOA Composite Application

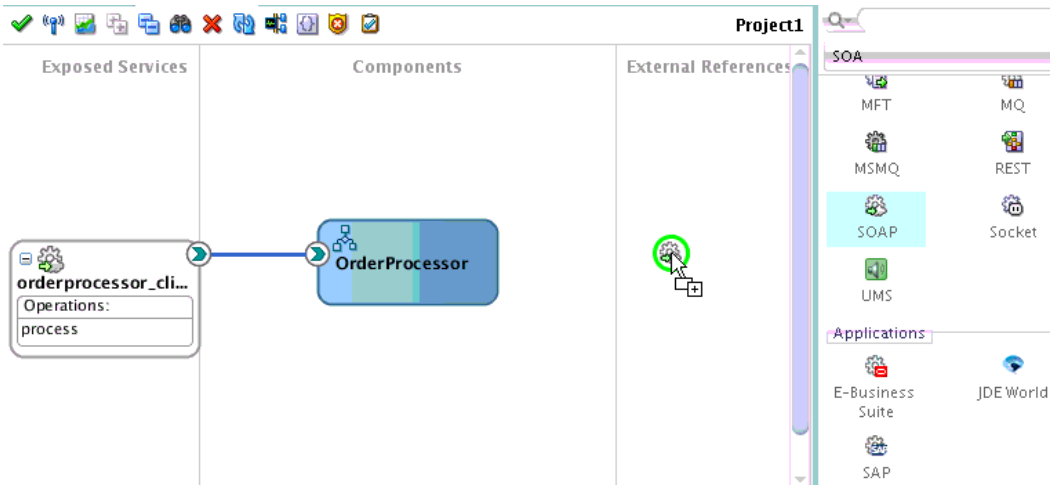
To integrate a binding component in a SOA composite application:

1. From the **Technology** section of the Components window, 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. Not all adapters can be dropped in both swimlanes. If an adapter is only available for references, then you cannot drop it into the services swimlane.

   - 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 35-6 shows a SOAP web service being dragged into the composite. This action invokes a dialog for specifying various configuration properties.

Figure 35-6 Integration of a Web Service Binding Component into a Composite



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

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

If a SOA composite application uses a web service binding to define an endpoint reference, the composite cannot be invoked from a JSP/Java class. Web services binding is defined with the `binding.ws port="" location=""` tag in the `composite.xml` file. The following example provides details:

```
<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 for SOA composite interaction with ADF-BC Web Application. After deployment of a composite with ADF binding, invocation from a JSP/Java class is successful. The following example provides details:

```
<reference name="ADFWebService"
             ui:wsdlLocation="ADFWebService.wsdl">
    <interface.wsdl interface="http://example.com/hr/#wsdl.interface(HRAppService)"/>
    <binding.adf serviceName="{http://example.com/hr/}HRAppService"
                 registryName="hrapp_JBOServiceRegistry"/>
  </reference>
```

For this example, `hrapp` is the ADF-BC web application name.

Creating Tokens for Use in the Binding URLs of External References

You can create tokens in Oracle JDeveloper for the HTTP protocol, host, and port values in the binding URLs of external references. The values that you assign to the tokens are then substituted in place of the hardcoded HTTP host and port values in the `location` attribute of the `binding.ws` element of the `composite.xml` file.

For example, the following code shows the `location` attribute with hardcoded values for protocol (`http`), host (`host.us.example`), and port (`80`).

```
<binding.ws
port="http://www.globalcompany.example.com/ns/CreditAuthorizationService#wsdl.
endpoint(CreditAuthorizationService/CreditAuthorizationPort)"
location="http://host.us.example:80/apps/FusionOrderDemoShared/services/
creditAuthorization/CreditAuthorizationService.wsdl">
```

The following example shows the `location` attribute after the creation of tokens.

```
<binding.ws
port="http://www.globalcompany.example.com/ns/CreditAuthorizationService#wsdl.
endpoint(CreditAuthorizationService/CreditAuthorizationPort)"
location="${protocol}://${host1}:${port1}/apps/FusionOrderDemoShared/services/
creditAuthorization/CreditAuthorizationService.wsdl">
```

Note:

- You can *only* use tokens in the `location` attribute of the `binding.ws` element of the `composite.xml` file.
- You cannot use tokens for the protocol, host, and port values in other files, such as WSDL files, schema files, and so on.
- Oracle JDeveloper only updates token files on the local file system that include the token values. If you use a local token file at design time, you must move the tokens to the SOA server at runtime. For information about creating tokens during runtime, see *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

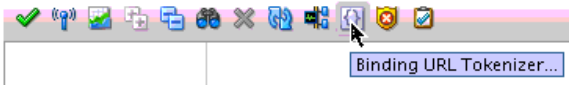
How to Create Tokens for Use in the Binding URLs of External References

Follow the steps in this section to create tokens for use in the binding URLs of external references.

To create tokens for use in the binding URLs of external references:

1. In Oracle JDeveloper, access the SOA composite application in which to create tokens.
2. Above the SOA Composite Editor, click the **Binding URL Tokenizer** icon. Figure 35-7 provides details.

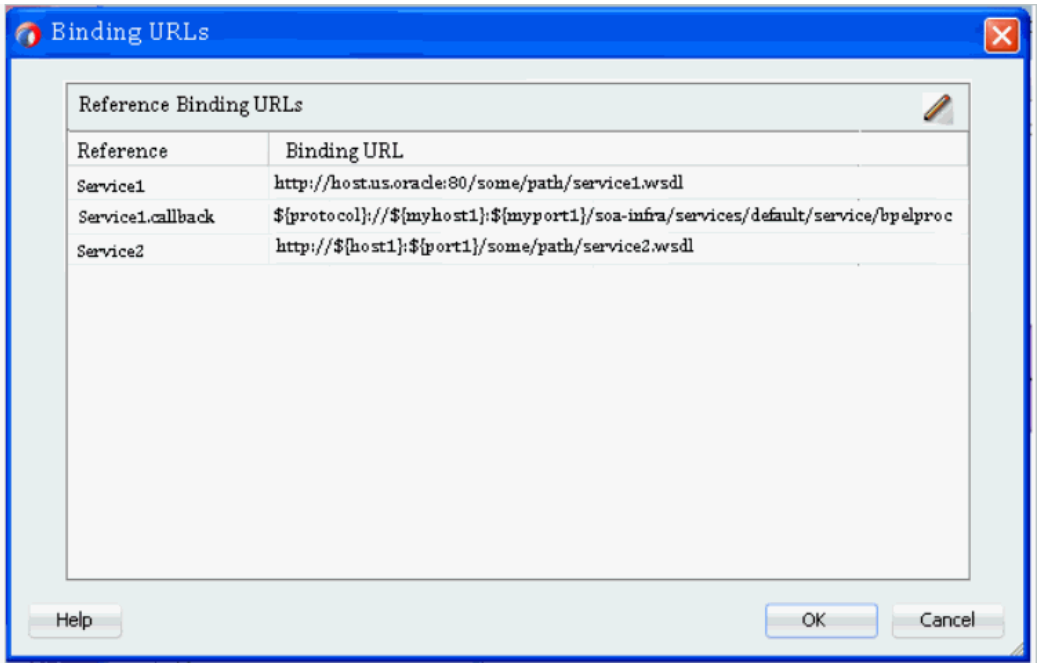
Figure 35-7 Binding URL Tokenizer Icon



The Binding URLs dialog appears, as shown in Figure 35-8.

- Binding URLs of each external reference that has a `binding.ws` element with a `location` attribute in the `composite.xml` file that starts with the following entries are automatically displayed:
  - `http`
  - `https`
  - `${` (for a URL that uses tokens in place of the hardcoded HTTP protocol, host, or port values)
  - `callbackServerURL`
- Binding URLs for REST references with the `location` attribute of the `binding.rest` element are automatically displayed.

Figure 35-8 Binding URLs Dialog



The **Service2** reference in Figure 35-8 also includes an override of the callback location using a reference property such as `callbackServerURL`:

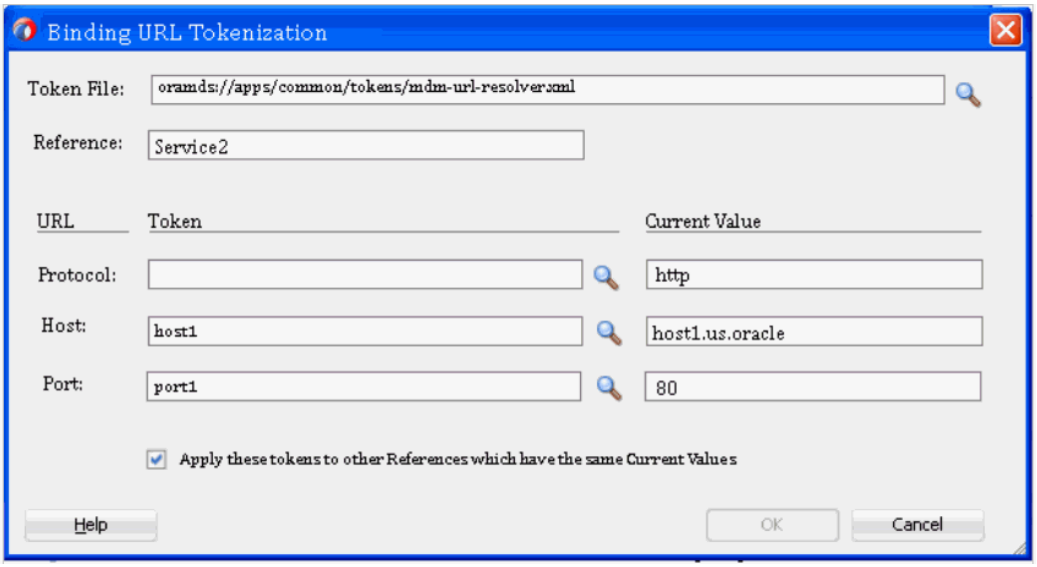
```
<property name="callbackServerURL" type="xs:string" many="false">
${protocol}://${myhost1}:${myport1}/soa-infra/services/default/service/
bpelprocess1_client_ep</property>
```

The `callbackServerURL` property can be tokenized as shown in Figure 35-8.

3. Double-click a row or select the row and click the **Edit** icon to create tokens for the HTTP protocol, host, and port values in the binding URLs of external references.

   The Binding URL Tokenization dialog appears, as shown in Figure 35-9.

Figure 35-9 Binding URL Tokenization Dialog



4. Provide values appropriate to your environment, as described in Table 35-4, and click **OK**.

Table 35-4 Binding URL Tokenization Dialog

Field	Description

Token File	Perform either of the following options:
	• Click the **Browse** button to access a dialog for selecting the token file that includes the token names and values. The file can be on the local file system. The names and values specified in this file replace the hardcoded names and values for protocol, host, and port in the `binding.ws` element. This field is automatically populated with your file selection on subsequent invocations of this dialog. If you specify a token file from the file system, it must be an XML file that follows this format:
	`<?xml version="1.0" encoding="UTF-8"?>`
	`<!DOCTYPE properties SYSTEM`
	`"http://java.sun.com/dtd/properties.dtd">`
	`<properties>`
	`<comment>`
	`URL Resolver file used by the Metadata`
	`manager to resolve $<variable> in URLs`
	`</comment>`
	`<entry key="protocol">oramds</entry>`
	`<entry key="host">MyHost</entry>`
	`<entry key="port">80</entry>`
	`</properties>`
	• Skip this field entirely if you want to manually enter new token names and values in the **Token** and **Current Values** fields, respectively.
	Tokens that are not saved to a file are only placed in the `location` attribute of the `binding.ws` element in the composite. It is expected that you supply a token file at runtime that has tokens matching those manually entered at design time.

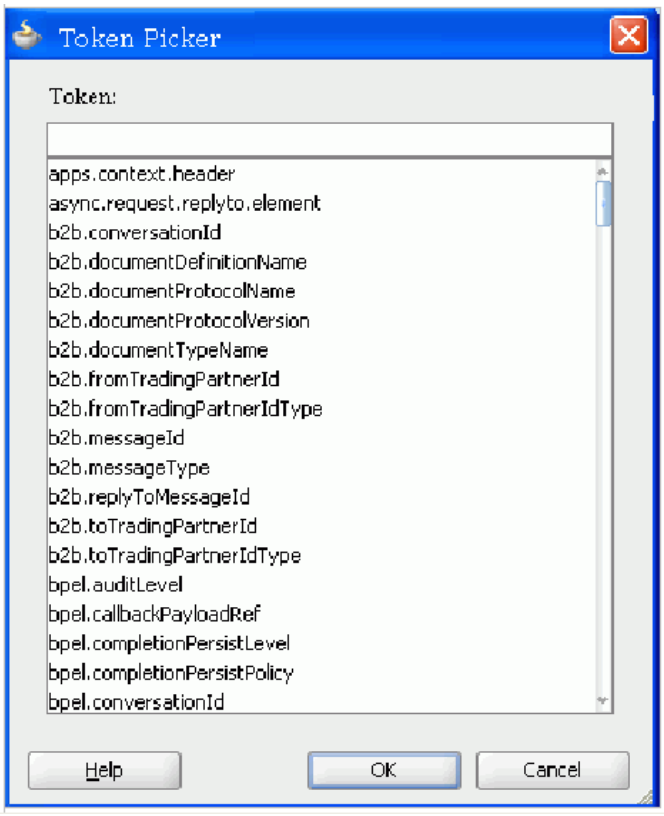
Reference	Displays the external reference you selected in Step 3.
Protocol	Displays the field in which to specify the protocol token name. • Click the **Browse** icon to select the token name to use from the Token Picker dialog. The Token Picker dialog is populated with the token names that appear in the token file you imported in the **Token File** field. The token name you select (for example, **port1**) and its default value (for example, **80**) are added to the **Token** and **Current Value** fields, respectively. If the token file is writable (meaning an unprotected file in the file system), you can change the current value of the token name. See Step 5 for details about accessing the Token Picker dialog. If the file is read-only, you are warned with a message and allowed to cancel the operation and continue. • Manually enter the token name and value to use. You can manually enter information in these fields regardless of whether you imported a file in the **Token File** field. If you imported a file that is writable in the **Token File** field and manually enter a token name, it is added to the file if it does not already exist. The current value for the new token name defaults to the value in the URL that is being tokenized.
Host	Displays the field in which to specify the host token name. See the description of the **Protocol** field for details about how to specify information.
Port	Displays the field in which to specify the port token name. See the description of the **Protocol** field for details about how to specify information.

Table 35-4 (Cont.) Binding URL Tokenization Dialog

Field	Description
Apply these tokens to other References which have the same Current Values	Deselect this check box if you do not want other external references with the same protocol, host, and port values to be replaced with the same tokens. If this check box is selected and you tokenize just one or two of the URL objects, then the references for only those objects are modified. For example, if you only tokenize the host (with a current value of **host1.us.oracle**), all references that have that same host value are updated.

5. If you selected the **Browse** button in the **Protocol**, **Host**, or **Port** fields, the Token Picker is displayed, as shown in Figure 35-10. This dialog lists all the tokens that you have defined in the file imported in the **Token File** field of the Binding URL Tokenization dialog.

Figure 35-10 Token Picker Dialog



6. Select the token name to use through one of the following options:

   - Scroll through the list and select the token.
   - Begin entering the name in the **Token** field until the name is automatically completed and the token is selected in the list.

7. Click **OK**.

   You are returned to the Binding URL Tokenization dialog with the selected token name and value displayed in the **Token** and **Current Value** fields, respectively.

36

Integrating REST Operations in SOA Composite Applications

This chapter describes how to integrate Representational State Transfer (REST) operations as service binding components and reference binding components in SOA composite applications. It also describes how to use a Web Application Description Language (WADL) file during reference binding component configuration.

This chapter includes the following sections:

For more information about using a REST adapter, see "Defining a Shipping Resource with a REST Service" of *Understanding Oracle SOA Suite*.

Introduction to REST Support

REST is an architecture for designing network applications. RESTful applications use HTTP requests to post data (create and update), get data (for example, make queries), update data, and delete data. REST provides an alternative to using web services.

Oracle SOA Suite provides the following REST support:

- Support in SOA composite applications:
  - Enable REST support in new or existing services.
  - Integrate with external REST APIs.
  - Orchestrate a set of RESTful state transitions (RPC/Hypermedia as the Engine of Application State (HATEOAS) approach).
  - Support for XML, JavaScript Object Notation (JSON) (with automatic translation to and from XML), URI sample, and URL-encoded GET/POST data.
  - Generation of sample URI for REST service operations.
  - Support for WADL services. The WADL can be provided by a deployed Oracle SOA Suite or Oracle Service Bus service or a non-Oracle SOA Suite or Oracle Service Bus service such as a Jersey REST service.
- Ease of development:

  - Oracle JDeveloper wizard provides several options for modeling REST interfaces and WSDL operation bindings:
    - Manually define resource paths and REST operations to generate an underlying WSDL that contains the mapping from the REST definition to the WSDL.
    - Select the WSDL of the service component or external reference from which to map WSDL operations to resource paths and HTTP verbs.
    - Select a WSDL from many sources (for example, the application server or SOA-MDS) from which to automatically populate the REST adapter with operation mappings.
  - Readable API that publishes each method used upon deployment.
  - Ability to browse and consume Oracle REST endpoints (including Oracle Service Bus) from within Oracle JDeveloper.
- Oracle Web Service Manager (OWSM) policy support for REST security.
- Support for the following use cases:
  - Get a list of customers
  - Create a new customer
  - Get customer details
  - Update customer details
  - Delete a customer
  - Create a new address for a customer
  - Get an address of a customer
  - Update the address of a customer

Creating REST Support in Service and Reference Binding Components

Oracle SOA Suite components, services, and reference can be selected and exposed as a REST service. This section describes how to perform the following tasks:

- Add REST support as a service binding component in an existing SOA composite application
- Add REST support as a reference binding component that can be invoked from a SOA composite application
- Configure REST support through shortcuts
- Generate REST schemas
- Use global token variables for host, port, and protocol values

Note:

- You cannot attach a REST binding to an asynchronous component (for example, an asynchronous BPEL process). If you attempt this attachment, a message is displayed that indicates this is not supported and suggests a workaround of placing an Oracle Mediator between the REST adapter and the service that has a one-way interface and routing the service callback to another (outbound) REST adapter service.
- You cannot connect a REST service binding component to a REST reference binding component.

How to Configure the REST Adapter as a Service Binding Component in a SOA Composite Application

This section describes how to add REST support as a service binding component that connects to an Oracle Mediator or BPEL process service component. For this example, a packing and shipping service expects a shipping resource that includes information about packing and shipping an order. A shipping resource is also returned with an updated order status.

To configure the REST adapter as a service binding component in a SOA composite application:

1. Right-click the **Exposed Services** swimlane in the SOA Composite Editor, and select **Insert** > **REST**. This action adds REST support as a service binding component to interact with the appropriate service component.

   The Create REST Binding dialog is displayed. Table 36-1 describes the fields of this dialog.

Table 36-1 Create REST Binding Dialog

Field	Description
Name	Enter a name for the REST service.
Type	Displays the type of REST service to create (service binding component or reference binding component).
Description	Enter a description. The description is published as part of the readable API used during deployment. The description is only available for service binding components.
Enforce XML Schema Ordering	Select to enforce the ordering of the XML schema. When selected, this reorders JSON payloads to match the order of elements in the XML schema. This includes inbound request payloads and responses from outbound requests. This option may add a performance overload. Selecting this check box sets the REST service binding property `reorderJsonAsPerXmlSchema` to `true` in the `composite.xml` file.

Table 36-1 (Cont.) Create REST Binding Dialog

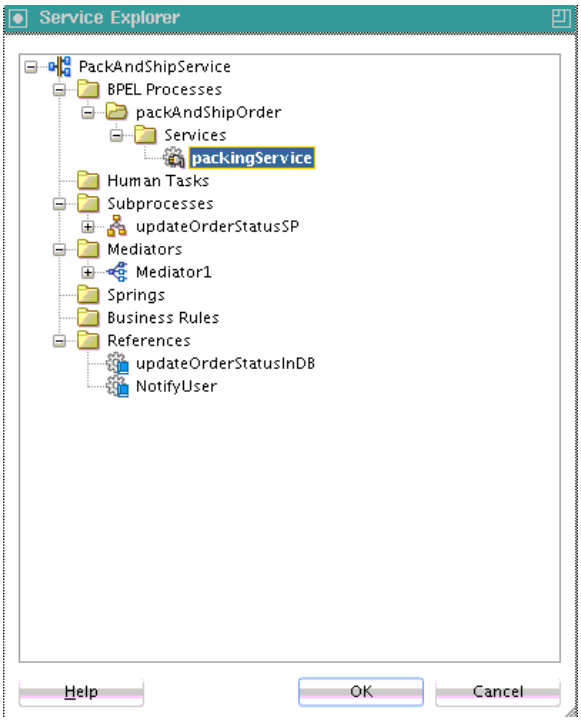
Field	Description
Resource Path	Double-click the default resources path (/) to update the resource path (this is described in Step 4) or click **Add** to add a new resource path.
Operation Bindings	Click **Add** and select to define the REST operations based on the WSDL of the component with which the REST service interfaces. • Select **Add operation binding** to manually define the rest service definition. You can define resource paths and REST operations. An underlying WSDL is generated that contains the mapping from the REST definition to the WSDL. For more information about this selection, see How to Generate Schemas Manually. • Select **REST enable component or reference** to invoke the Service Explorer dialog to select the WSDL of the service component or external reference from which to map WSDL operations to resource paths and HTTP verbs. This option is only available to service binding components. This selection is described in subsequent steps of this section. • Select **REST enable external webservice** to invoke the WSDL Chooser dialog to select a WSDL from many sources including the application server and SOA-MDS. After the WSDL is selected, the REST adapter is automatically populated with operation mappings for the selected WSDL. On completion of the dialog, a REST service, SOAP reference, and wire between the two are created.

2. In the **Operation Bindings** section, click the **Add** icon and select **REST enable component or reference**.

   The Service Explorer dialog is displayed.

3. Expand the navigator to select the WSDL of the service component (for this example, a BPEL process), and click **OK**. This action enables you to map WSDL operations to resource paths and HTTP verbs. Figure 36-1 provides details.

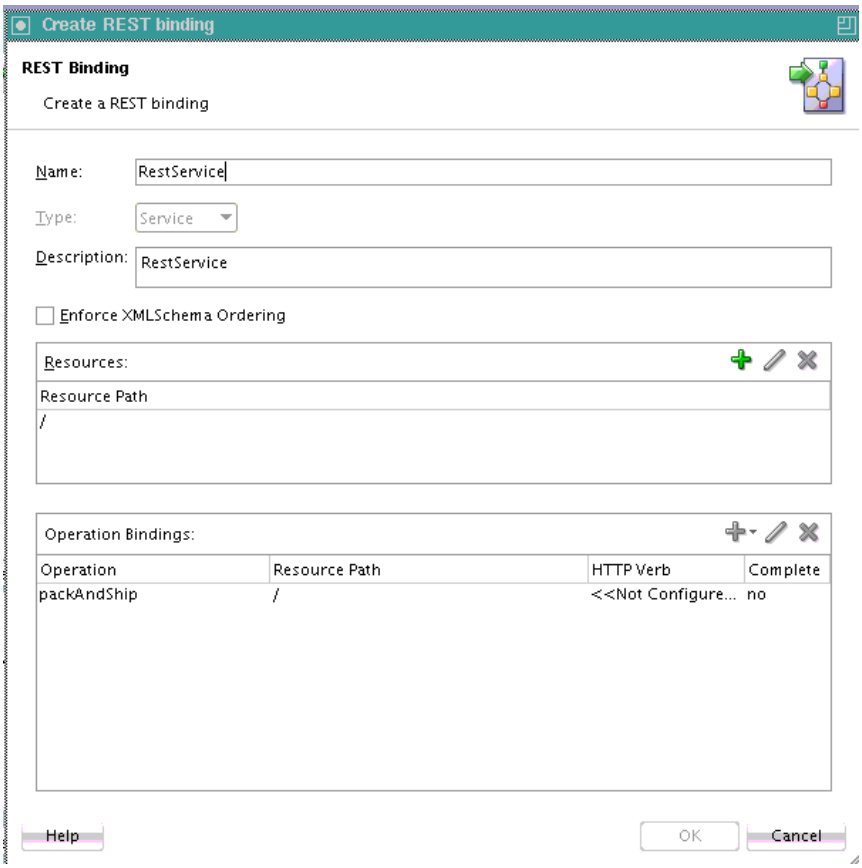
Figure 36-1 WSDL Selected to Map REST Operations to Resource Paths and HTTP Verbs



The Create REST Binding dialog is updated to appear as shown in Figure 36-2.

The selected WSDL is read and the WSDL operation **packAndShip** is mapped to resource paths and HTTP verbs in the **Operation Bindings** section. Note that the **Resource Path** and **HTTP Verb** sections require additional configuration mapping. This is also indicated by the value of **no** in the **Complete** column.

Figure 36-2 Create REST Binding Dialog



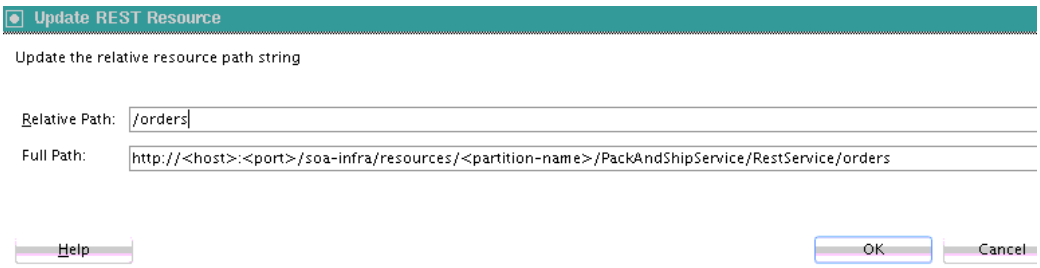
The resource path and HTTP verb for each of the operations now require configuration. For this example, there is only one operation. Depending upon your WSDL, multiple operations can be displayed in the **Operation** column in Figure 36-2.

4. In the **Resource Path** table of the **Resources** section, double-click the default path entry of **/**. You can also define the resource path before starting the operation bindings. In this case, the selected resource is used for the new bindings.

   This invokes the Update REST Resource dialog.

5. In the **Relative Path** field, enter the resource path (for this example, `/orders`), and click **OK**. Figure 36-3 provides details.

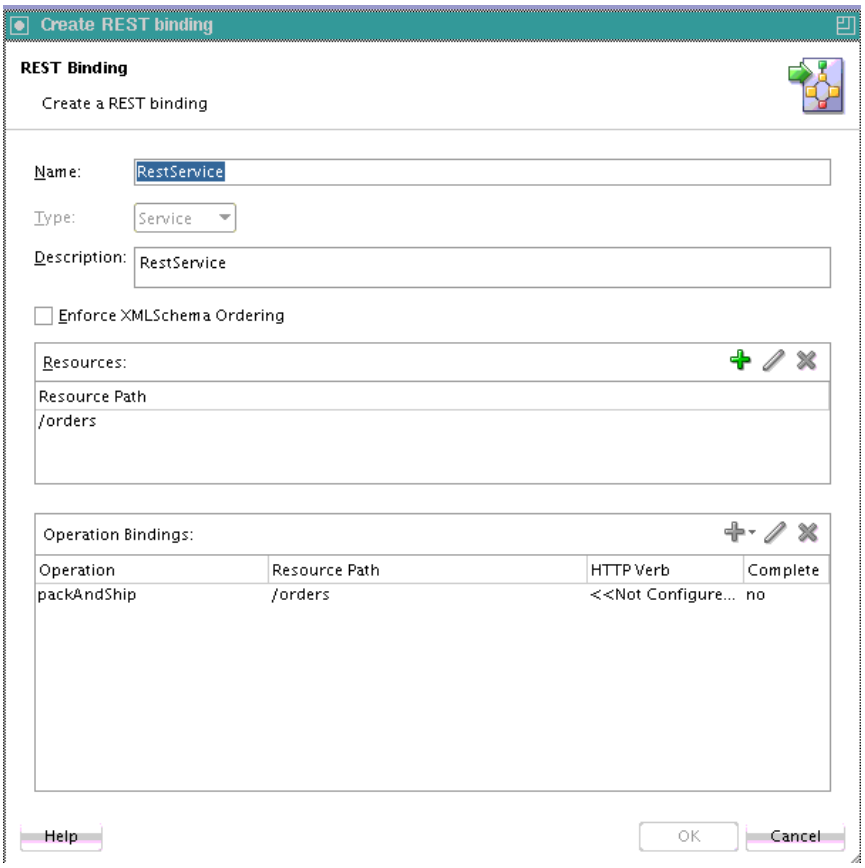
Figure 36-3 Update REST Resource Dialog



Operation mappings that have the old resource path are updated with the new resource path in the **Resources** section and **Operation Bindings** section (for this

example, **/orders**). If you are updating an existing component, all operations are typically updated. Figure 36-4 provides details.

Figure 36-4 Resource Path Updated in Resources and Operation Bindings Sections

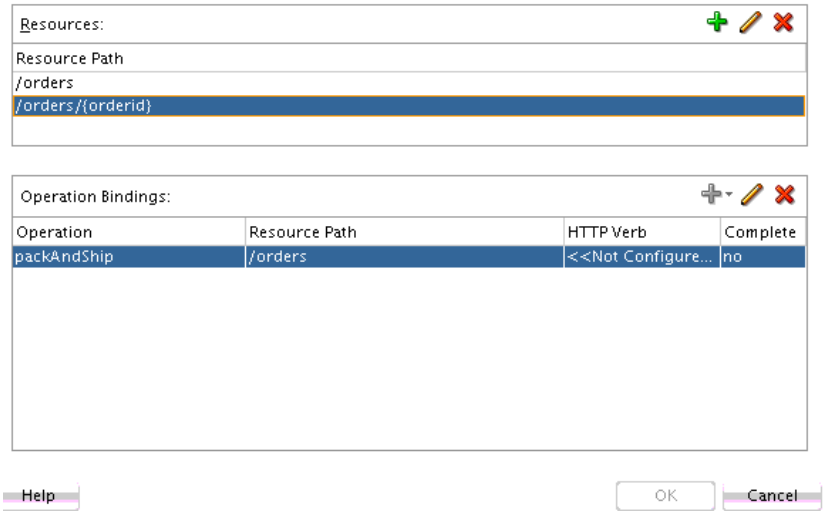


You can create additional resource paths as needed by clicking the **Add** icon in the **Resources** section to display the Create REST Resource dialog.

The **HTTP Verb** column of the **Operation Bindings** section now requires configuration.

6. In the **Operation Bindings** section, select an operation and click **Edit** or click the row (for this example, **packAndShip**). Figure 36-5 provides details.

Figure 36-5 Operation Bindings Selection



The REST Operation Binding dialog is displayed.

This dialog enables you to select the HTTP verb for the operation and populate the **URI Parameters** section in order to bind an HTTP verb and resource to a WSDL operation and map REST parameters to the WSDL schema of the component service. Table 36-2 describes the fields of this dialog.

Table 36-2 REST Operation Binding Dialog

Field	Description
Operation	Displays the WSDL operation name that is being mapped. In most cases, you can specify a name that is used in the generated WSDL. • If you selected **Add operation from component service** in the Create REST Binding dialog and then selected the WSDL file in the Service Explorer dialog, this field is read-only. • If you selected **Add operation binding** in the Create REST Binding dialog, you can edit this field.
Resource	Select an existing URL resource path from the list (this is described in Step 7) or click the **Add** icon to add a new resource path. The selected resource path is added to the **URI Parameters** table of the **Request** section at the bottom of this dialog. If the selected resource contains a template variable, such as **{var}**, the variable is added to the URI parameters.
HTTP Verb	Select the verb to be bound to the WSDL operation (for example, **GET**, **PUT**, **POST**, or **DELETE**). (This is described in Step 8.) This selection populates the **URI Parameters** table with mappings from the incoming REST query parameters to the WSDL schema. The HTTP verb for the operation is also added to the **Operation Bindings** section of the Create REST Binding dialog after clicking **OK**.
Description	Enter an optional description. Reference binding components have the **Base URI** field.

Table 36-2 (Cont.) REST Operation Binding Dialog

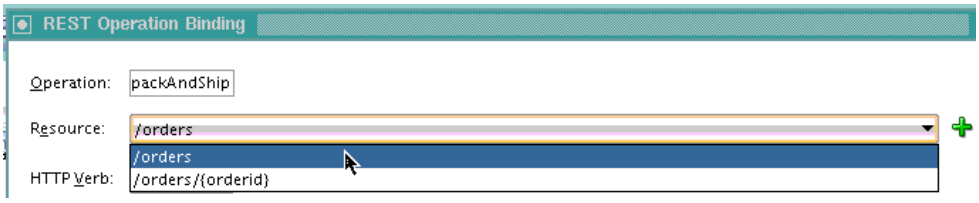
Field	Description
Request	The **Schema** section displays the request schema being used. • **Schema URL**: Displays the request schema to use. If you selected **REST enable component or reference** in the Create REST Binding dialog, this field is read-only because the schema is obtained from the service's operation WSDL. If you selected **Add operation binding** in the Create REST Binding dialog, you must browse for a schema or create a new schema from a sample using the Native Format Builder wizard. • **Element**: Displays the element to use. The **Payload** section enables you to specify the format of the request payload: XML (default selection), JSON, URL-encoded, or no payload. Click **Generate Sample Payload** to view a sample of the selected request payload. The **URI Parameters** section enables you to specify the mapping from the REST query parameters to the WSDL schema. This section is automatically populated when a schema is specified (depending on the verb) in the **HTTP Verb** list. For **GET** and **DELETE** verbs, the parameters are bound to the WSDL schema. For **POST** and **PUT** verbs, the inbound payload is mapped to the WSDL schema. **a.** Click the **Generate Sample URL for operation** icon (first icon) to generate a sample URL based on all previously entered binding information. This option is typically selected after all parameters are configured. **b.** Click the **Add parameter** icon (second icon) to manually add a mapping parameter. **c.** In the **Style** column, click a specific row to invoke a list that enables you to select **query** or **template**. Template variables are typically used for **POST** and **PUT** operations. Query parameters are typically used for **GET** and **Delete** operations. **d.** In the **Type** column, select the data type of the parameter. All XSD primitive types are supported. In most case, when the parameter is automatically generated from a schema, the type is already set for you. If you create a new parameter, the **Type** column enables you to select the type from the list. **e.** In the **Default Value** column, you can set a default value at design time for a parameter. If a URI parameter is missing in the REST request, the corresponding default value is used by the REST service. **f.** In the **Expression** column, click a specific row to invoke the Expression Builder dialog for adding an XPath expression function. If there is no schema defined for an operation, the Expression Builder parameter only shows property variables and no message variables. The expression binds a parameter to a field in the WSDL schema. In all cases in which the parameter has been automatically generated (based on an existing or generated schema), this expression is already generated for you. You only add an expression if you want to add a new parameter and bind it to something else,

Table 36-2 (Cont.) REST Operation Binding Dialog

Field	Description
	such as a runtime property. The XPath expression specifies the location in which to insert the particular URI parameter in the normalized message.
Response	The **HTTP Statuses** section enables you to specify the HTTP status code. You can enter multiple statuses, separated by spaces. For a reference, these are the possible statuses that are interpreted as successful. For a service, these are the possible successful statuses that can be returned (as set by a service component such as BPEL). The **Payload** section enables you to specify the possible response payloads: XML (default), JSON, or no payload. The output returned at runtime depends on the incoming requests. Click **Generate Sample Payload** to view a sample of the selected response payload. The **Schema** section displays the response schema being used if a possible payload type has been selected. If no payload has been selected, this field is not displayed. • **Schema URL**: Displays the response schema to use. If you have not specified a schema, you can select to browse for an existing schema or create a new schema from a sample with the Native Format Builder wizard. This wizard enables you to create a schema from a JSON interchange format, XML sample, URI-encoded format, or URI sample. For more information, see How to Generate Schemas from Samples. • **Element**: Displays the element to use. The **Fault Bindings** section displays the response fault name, type, status, and schema. If fault details are defined in the WSDL file, a fault binding is automatically created in this section. You can also manually define fault bindings.by clicking the **Add** icon. For more information about faults, see What You May Need to Know About REST Fault Binding.

7. From the **Resource** list, select the new resource, as needed. Figure 36-6 provides details.

Figure 36-6 New Resource Selection

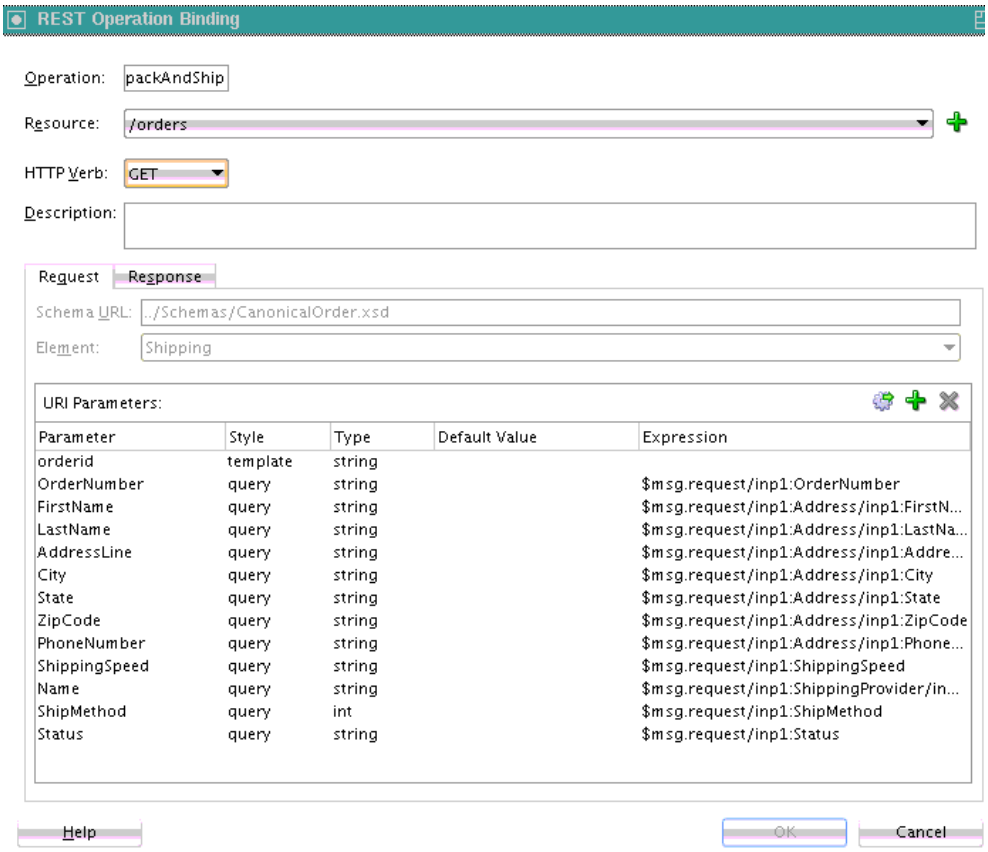


The **URI Parameters** section is updated with your selection.

8. From the **HTTP Verb** list, select the operation (for this example, **GET**).

The **URI Parameters** section is updated with your selection. Figure 36-7 provides details.

Figure 36-7 HTTP Verb Selection



The style (**query** or **template**) is automatically selected in the **Style** column of the **URI Parameters** section.

If you select or create a new REST resource that contains a template variable, Oracle JDeveloper attempts to create the template parameter with the same name. If a parameter with that name already exists, it is reused (and made into a template parameter if it was a query parameter). Duplicate parameter names are never created. You receive an error if a duplicate parameter is manually created.

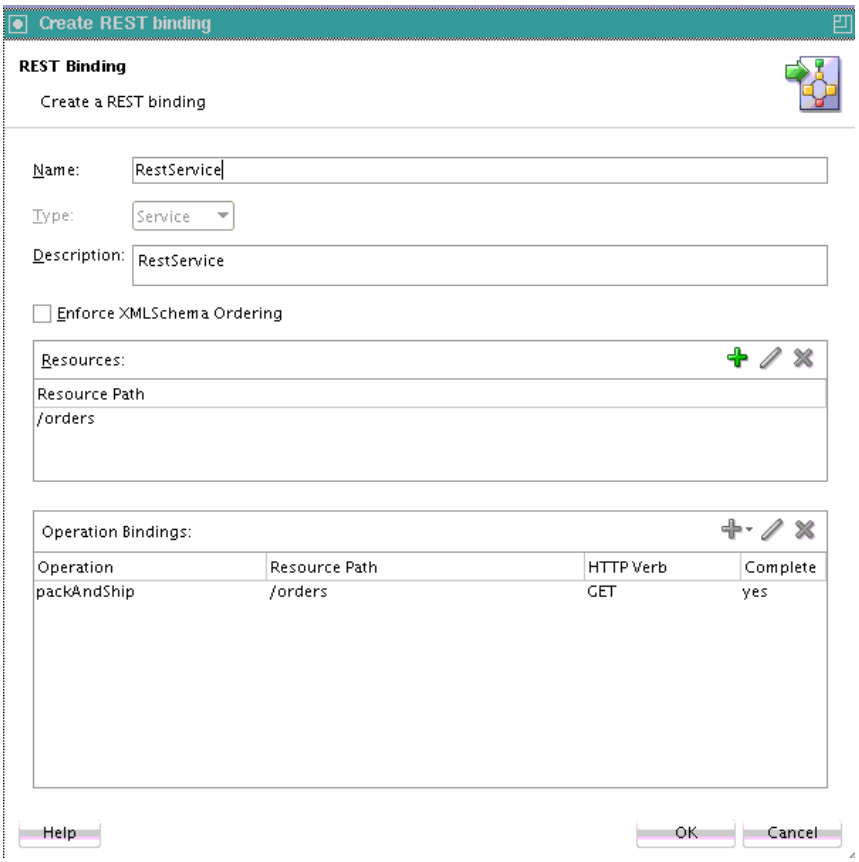
9. Click the **Response** tab to view HTTP status code, payload output type, schema, and fault binding details. Since the schema was already defined in this example, those sections are disabled from editing.

10. Double-click the fault name to invoke the REST Fault Binding dialog. For more details about this dialog, click the **Help** icon or see What You May Need to Know About REST Fault Binding.

11. Edit as necessary, and click **OK**.

12. Click **OK** to return to the Create REST Binding dialog, as shown in Figure 36-8. The HTTP verb you added is displayed.

Figure 36-8 Create REST Binding Dialog



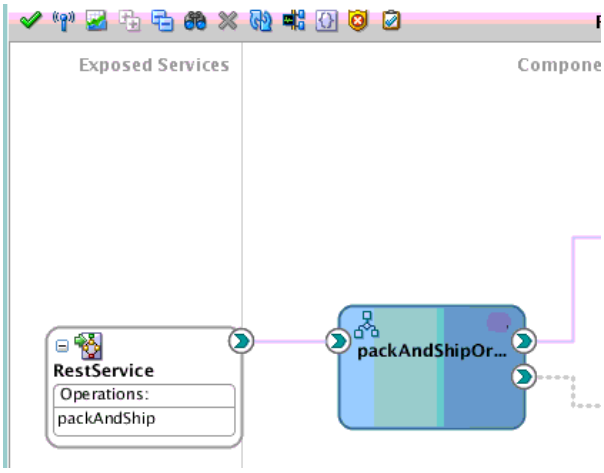
This example included only one operation. If your WSDL contained additional operations, they are displayed in the **Operation** column of the **Operation Bindings** section.

13. In the **Operation Bindings** section, select an operation and click **Edit** to define resources and HTTP verbs for any remaining operations.

14. Click **OK** to return to the SOA Composite Editor.

    The REST service is wired to the BPEL process service component. Figure 36-9 provides details.

Figure 36-9 REST Adapter Service Wired to BPEL Process



How to Configure the REST Adapter as a Reference Binding Component in a SOA Composite Application

This section describes how to add REST reference binding component to a SOA composite application. This example describes how to add an existing REST reference that is based on a WADL file. WADL files provide an XML description of HTTP-based web applications.

To configure the REST adapter as a reference binding component in a SOA composite application:

1. Right-click the **External References** swimlane of the SOA composite application, and select **Insert** > **REST** to invoke a REST reference from a service component such as a BPEL process or Oracle Mediator.

   The Create REST Binding dialog is displayed.

2. In the **Operation Bindings** section, click the **Add** icon and select **Add operations based on WADL service**.

   The WADL Location dialog is displayed.

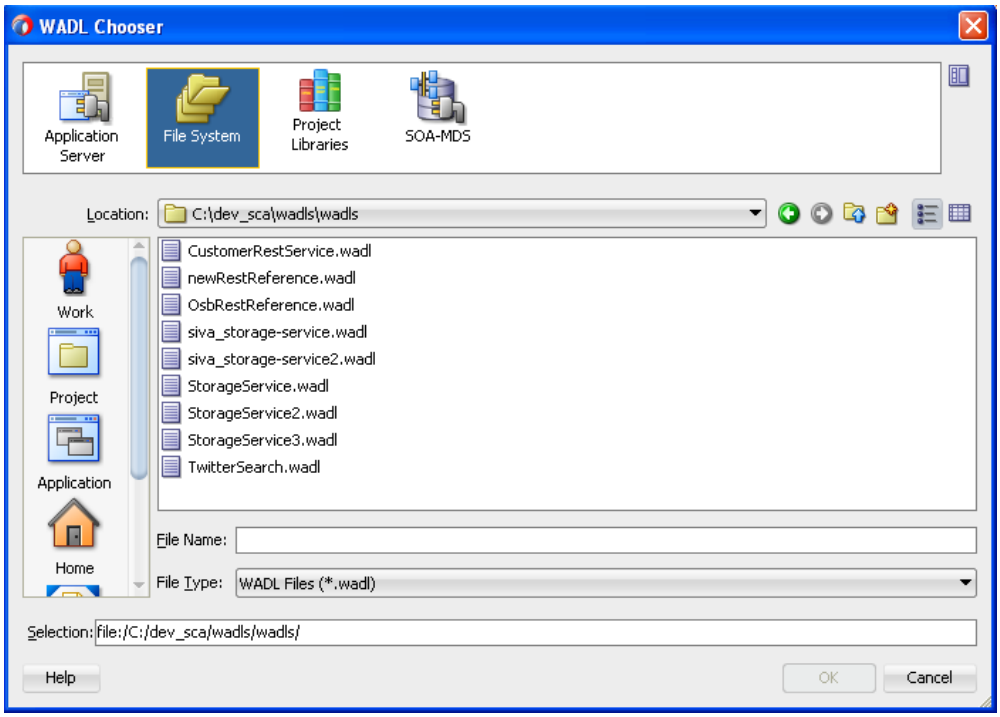
3. Specify a WADL file through one of the following methods:

   a. In the **WADL URL** field, specify the URL of the WADL file, then go to Step 4.

   or

   b. Click the **Search** icon to invoke the WADL Chooser dialog for selecting the WADL file. Options are provided for finding WADLs in the local file system or project, in the design-time Oracle Metadata Services Repository (MDS Repository), or by connecting to an application server to find WADLs associated with deployed Oracle SOA Suite or Oracle Service Bus services. Figure 36-10 provides details.

Figure 36-10 WADL Chooser Dialog



c. Select the WADL file, and click **OK**.

d. Go to Step 4.

4. Click **OK**.

5. Select **copy schema artifacts into the project** to copy schemas referenced in the WADL file to the local project because they are used by the new REST adapter reference. This is the recommended method.

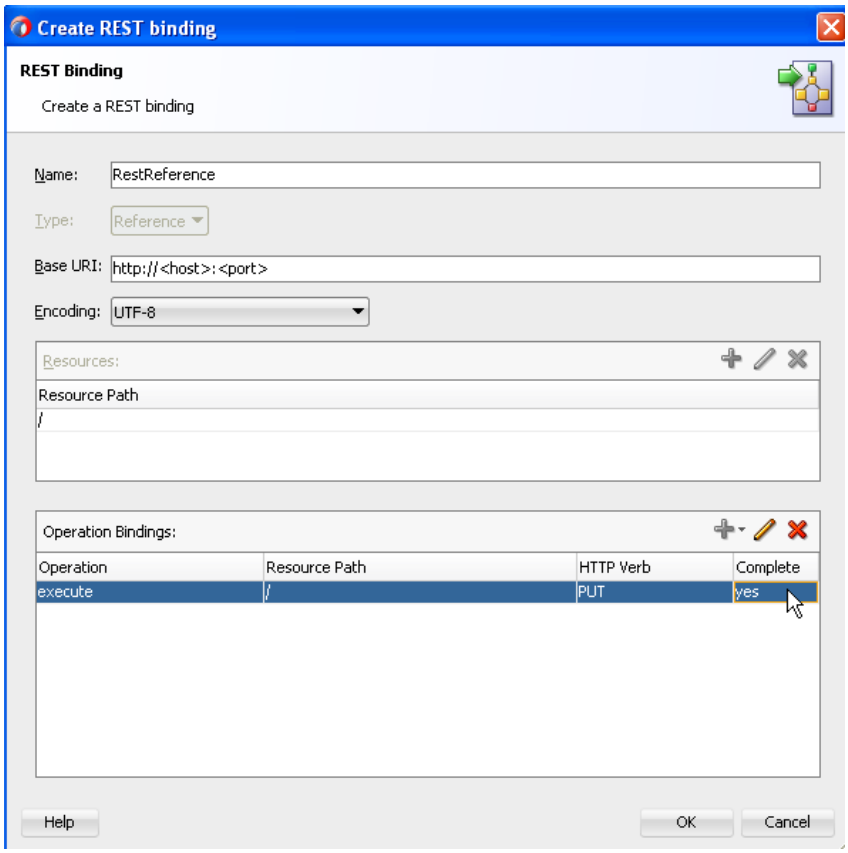
6. See the step based on the type of WADL file selected. Table 36-3 provides details.

Table 36-3 WADL File Status

If the Selected WADL File Was Provided By...	Then...	See Step...
An Oracle SOA Suite or Oracle Service Bus REST service	The Create REST Binding dialog is completely configured with information from the WADL file. All operations, resource paths, and verbs are displayed in the **Operation Bindings** section. A complete configuration is indicated by a value of **yes** in the **Complete** column of the **Operation Bindings** section.	6.a
A non-Oracle SOA Suite or Oracle Service Bus REST service such as a Jersey service.	The WADL Parsing Issues dialog indicates that additional configuration is required.	6.b

a. View the Create REST Binding dialog and its contents, and click **OK**. Figure 36-11 provides details.

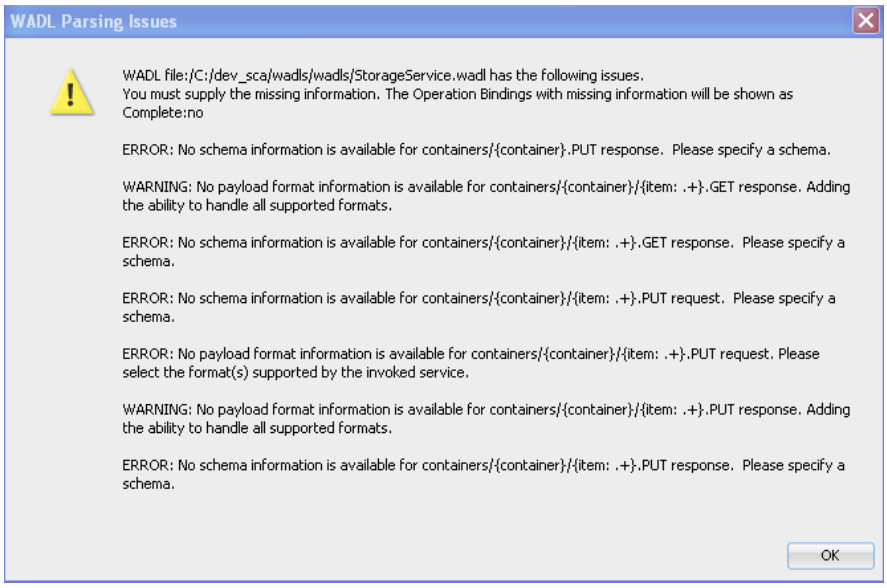
Figure 36-11 Create REST Binding Dialog with All Operational Bindings Complete



b. Review the list of recommended corrective actions in the WADL Parsing Issues dialog shown in Figure 36-12, then click **OK**. The same information is written to the Oracle JDeveloper Log window for later reference.

A WADL file for a non-Oracle SOA Suite or Oracle Service Bus service typically does not include all required information. You must manually complete configurations. In many cases, such as in Figure 36-12, the WADL file used does not supply the schemas required to bind the REST reference to a WSDL operation. The schemas can often be generated using the Native Format Builder wizard using a sample payload provided by the REST service provider.

Figure 36-12 WADL Parsing Issues



The Create REST Binding dialog is displayed.

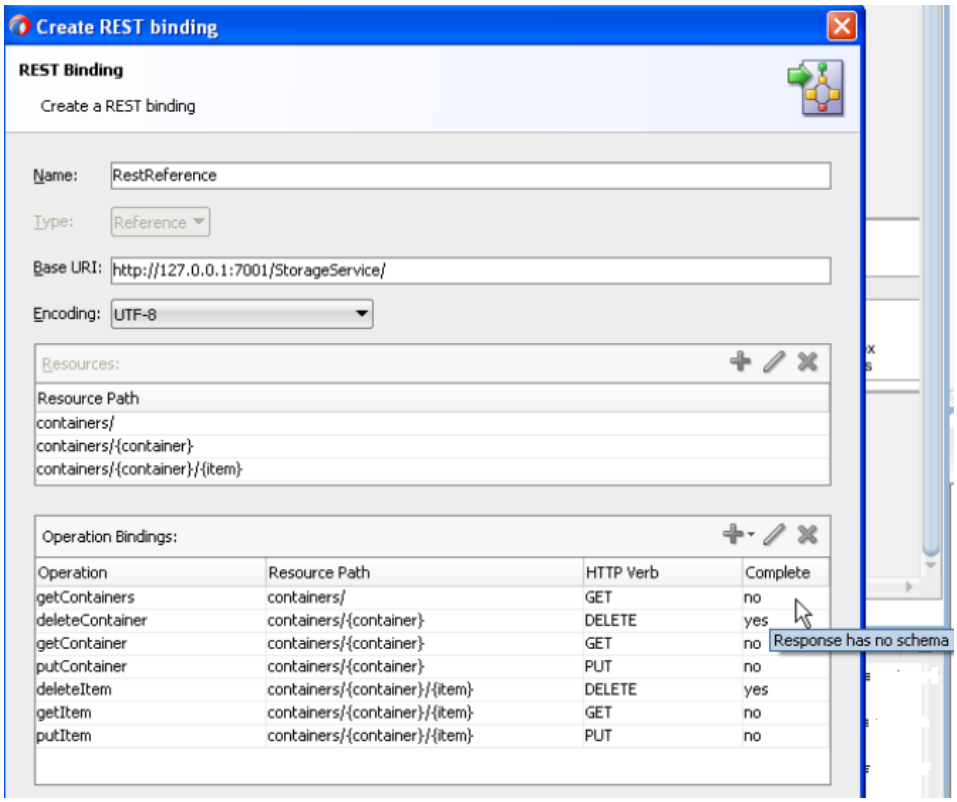
For a REST binding that requires further configuration, a value of **no** is displayed in the **Complete** column of the **Operation Bindings** section. Tool tips in the **Complete** column for each **no** value identify what is missing so that you can perform corrective actions. The **OK** button is disabled as long as at least one operation has a value of **no** in the **Complete** column. Figure 36-13 provides details.

Note:

All error and warnings are also displayed in more detail in the Log window in Oracle JDeveloper. For example, the Log window contains entries such as the following:

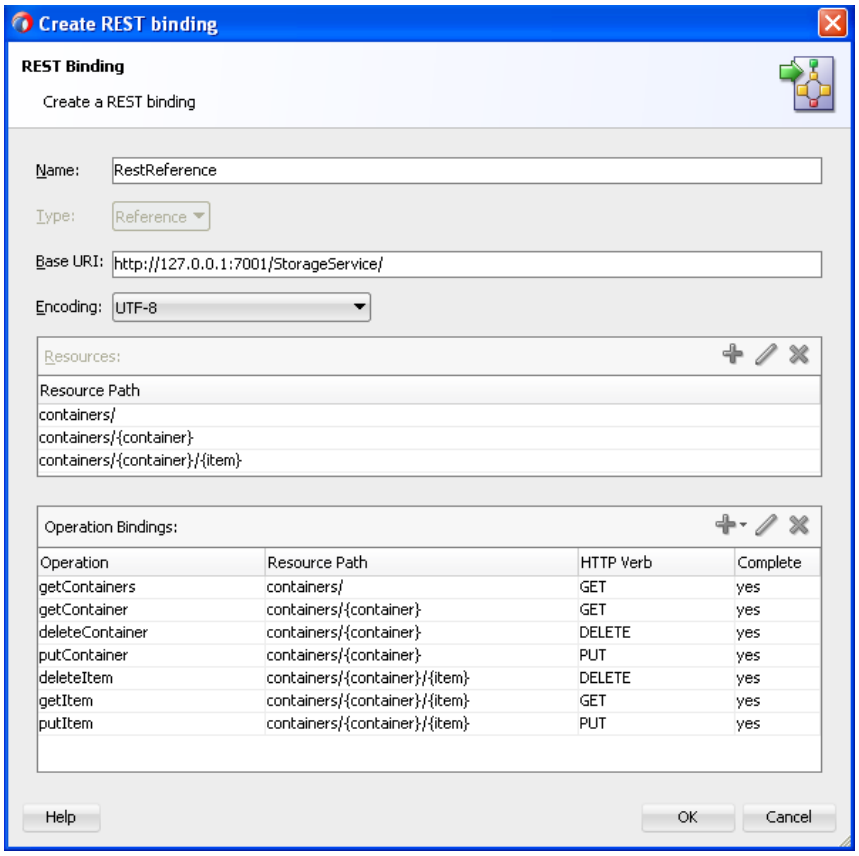
```
ERROR at [resource path: containers/{container}, method name: PUT,
request/response: response, representation mediaType:
application/xml ] - No schema information is available for
containers/{container}.PUT response.  Please specify a schema.
```

Figure 36-13 Create REST Binding Dialog with Incomplete Operational Bindings



c. Perform the corrective actions indicated by the tool tips. When configuration has been successfully completed, a value of **yes** is displayed for all operations in the **Complete** column of the **Operation Bindings** section. Figure 36-14 provides details.

Figure 36-14 Create REST Binding Dialog with All Operational Bindings Complete



For more information about the SOA design-time MDS Repository, see Managing Shared Data with the Design-Time .

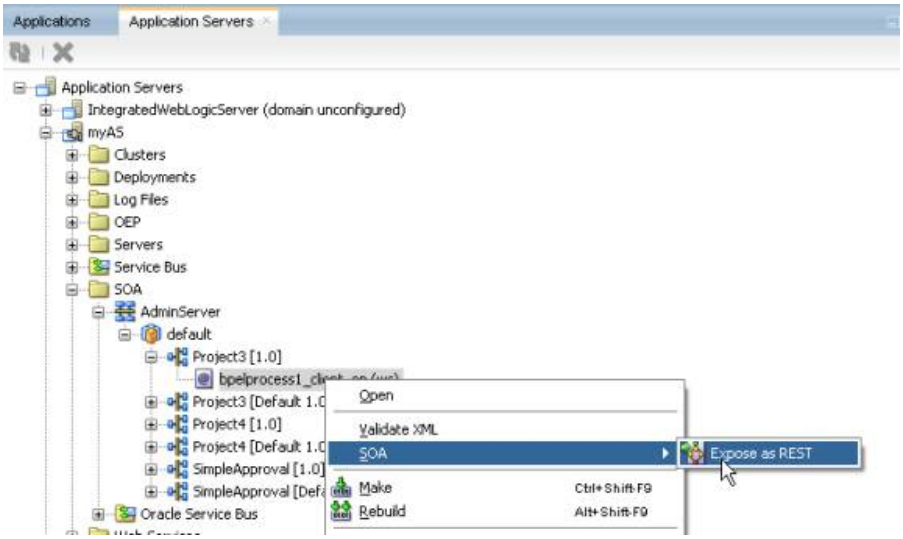
How to Configure the REST Adapter Through Shortcuts

You can configure the REST adapter through several shortcuts based on WSDL or WADL files.

To generate a REST service based on a web service deployed on an application server:

1. From the Oracle JDeveloper main menu, select **Window** > **Application Servers**.
2. Right-click a WSDL service and select **SOA** > **Expose as REST**. Figure 36-15 provides details.

Figure 36-15 Automatic REST Adapter Service Binding Component Configuration



The Create REST Binding dialog is invoked and prepopulated with operation mappings from the selected WSDL file.

3. Complete any necessary configuration by following the procedures in How to Configure the REST Adapter as a Service Binding Component in a SOA Composite Application.

Note:

You are prompted to make a local copy of the selected WSDL and its dependent artifacts. If you select to make a local copy, the `binding.ws` section for the SOAP reference binding component contains the original concrete WSDL/endpoint location that was selected and the copied WSDL is used as the abstract WSDL (in the composite import, and so on).

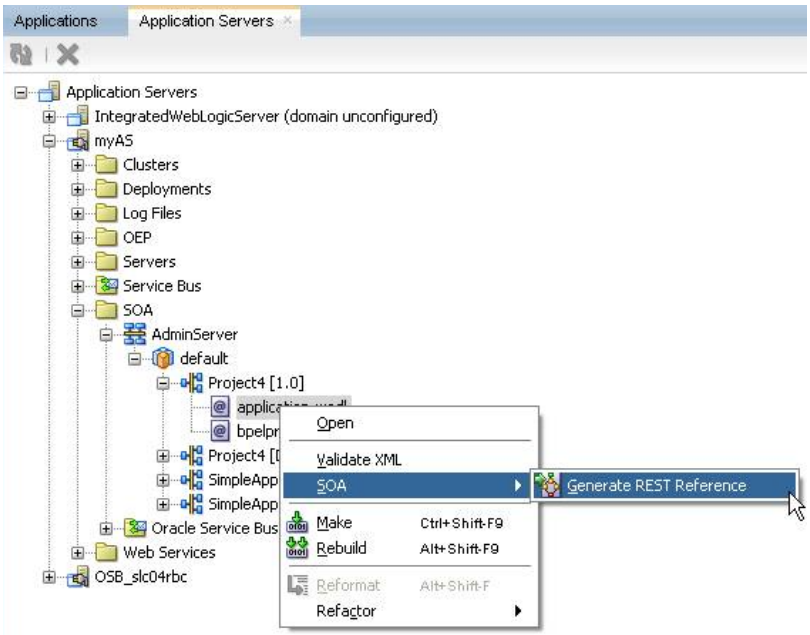
When configuration is complete, a REST service binding component is wired to a SOAP reference binding component. Figure 36-16 provides details.

Figure 36-16 REST Service Binding Component and SOAP Reference Binding Component Configuration

To generate a REST reference based on a REST service deployed on an application server:

1. From the Oracle JDeveloper main menu, select **Window** > **Application Servers**.
2. Right-click a REST/WADL service and select **SOA** > **Generate REST Reference**. Figure 36-17 provides details.

Figure 36-17 Automatic REST Adapter Reference Binding Component Configuration

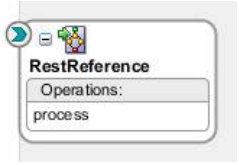


The Create REST Binding dialog is invoked and prepopulated with information from the selected WADL file.

3. Complete any necessary configuration by following the procedures in How to Configure the REST Adapter as a Reference Binding Component in a SOA Composite Application.

   When configuration is complete, a REST reference binding component is displayed, as shown in Figure 36-18.

Figure 36-18 REST Adapter Reference Component



To generate a REST service based on a SOA component's WSDL service:

1. In the SOA Composite Editor, right-click a SOA component service or an external reference, and select the **Expose as REST** option. Figure 36-19 provides details.

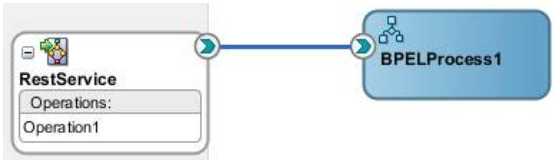
Figure 36-19 Expose as REST Option

The Create REST Binding dialog is invoked and prepopulated with information from the selected WADL file.

2. Complete any necessary REST adapter configuration.

   When configuration is complete, REST support (for this example, a service binding component) is created and automatically wired to the interface on which you clicked. Figure 36-20 provides details.

Figure 36-20 REST Adapter Service Binding Component



How to Generate Schemas Manually

If you do not have a schema or sample data to generate a schema, you can manually enter the parameters for which to generate a schema. To manually define a schema, click the **Add** icon, and select **Add operation binding** in the **Operation Bindings** section of the Create REST Binding dialog.

Note the following guidelines:

- In the **URI Parameters** section of the REST Operation Binding page, enter the necessary query or template parameters. You can select a data type for each parameter, but you do not need to enter an expression. If there is no schema and payload specified, when you click **OK**, the parameter schema is automatically generated (embedded in the WSDL). Parameters that have an expression that is mapped to a runtime property are not included in the generated schema.
- If there is no schema defined for an operation, the Expression Builder dialog that is accessible from the **Expression** column only shows property variables and no message variables.

For complete instructions about creating REST support, see How to Configure the REST Adapter as a Service Binding Component in a SOA Composite Application and How to Configure the REST Adapter as a Reference Binding Component in a SOA Composite Application.

How to Generate Schemas from Samples

You can generate schemas from sample files, including JSON interchange format.

1. In the **Request** section of the REST Operation Binding dialog, click the **Define Schema for Native Format** icon to the right of the **Schema URL** field. Figure 36-21 provides details.

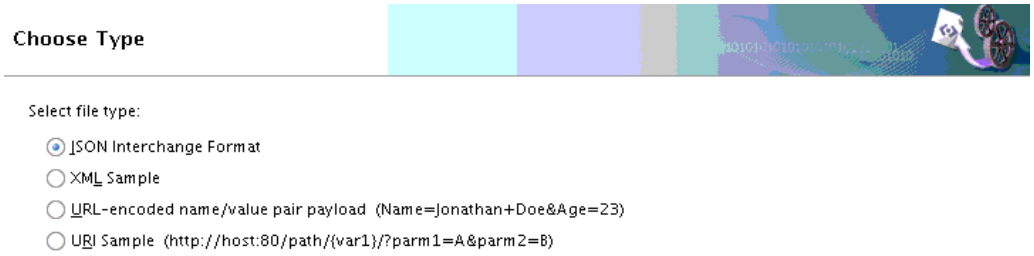
Figure 36-21 Define Schema for Native Format Icon

The Native Format Builder wizard is displayed.

2. Proceed through the initial pages of the wizard until you access the Choose Type page.

   This page enables you to select to generate schemas from different format types. Figure 36-22 provides details.

Figure 36-22 Types From Which to Generate Schemas



3. Select an appropriate type.

   Each type provides an area in which to paste a JSON, XML, or URL sample or select a sample file to import.

   For more information about JSON interchange formats, see What You May Need to Know About Converting a JSON Interchange Format to a REST Schema.

For complete instructions about creating REST support, see How to Configure the REST Adapter as a Service Binding Component in a SOA Composite Application and How to Configure the REST Adapter as a Reference Binding Component in a SOA Composite Application.

How to Use Global Token Variables

You can assign global token variables for the host name, port number, and protocol in the **Base URI** field of the Create REST Binding dialog.

To use global token variables:

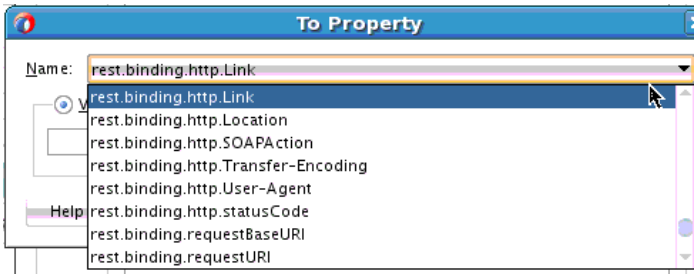
1. Above the SOA Composite Editor, click the **Binding URL Tokenizer** icon.
2. Select the REST external reference, and click the **Edit** icon.
3. Specify the `mdm-url-resolver.xml` file or manually enter values for host, port, and protocol (`http` or `https`), and click **OK**.

   The values that you assign to the tokens are then substituted in place of the hard-coded HTTP host and port values in the `location` attribute of the `binding.ws` element of the `composite.xml` file. For more information, see Creating Tokens for Use in the Binding URLs of External References.

How to Set REST Header Properties

Normalized message properties are available for certain standard HTTP headers. These properties are displayed for selection in the **Properties** tab of receive and reply activities in a BPEL process in Oracle JDeveloper, as shown in Figure 36-23. These header properties are not propagated by default across the service engines. You must manually propagate them by providing the appropriate assignment logic.

Figure 36-23 Normalized Message Properties in Receive and Reply Activities



Inbound and Outbound Headers

For inbound cases in which an external client is interacting with a SOA REST service, you can configure the service to send a hyperlink to the next resource with which the client interacts. This hyperlink can be sent in the payload of the response or the HTTP link response header. The following normalized message properties build and return the next link:

- **rest.binding.requestBaseURI**

  This property is available on the request message. It holds the base URI of the REST service. In the SOA composite application, this can be combined with the path of another resource in the same REST service to build an absolute URL linking back to that resource.

- **rest.binding.http.Link**

  When this property is set on the response message, a link header is added to the HTTP response. The value of this header is the value of the normalized message property.

For outbound cases in which the SOA composite application is invoking an external REST service, the service may return a response with the next link either in the link header or the payload. The following normalized message properties are available to help get the next link and invoke the resource located at that link:

- **rest.binding.http.Link**

  If the HTTP response from the external REST service contains a link header, a **rest.binding.http.Link** property is added to the response message. The value of this property is the value in the link header.

- **rest.binding.requestURI**

  The resource located at the next link can be invoked by setting the **rest.binding.requestURI** property on the request message. If set, the URL in this property overrides the URL provided at design time. It invokes the external REST service.

There are several preconditions that must be satisfied when a SOA composite application is invoking REST resources based on the next link it receives from the external REST service:

- The potential resources that can be invoked must be designed in the REST reference binding component at design time.
- The structure of the request and response must be known and modeled at design time.

For information about setting normalized message properties in the **Properties** tab, see Propagating Normalized Message Properties Through Message Headers.

Custom Header Support

REST services and references are capable of handling custom HTTP headers. Table 36-4 provides details.

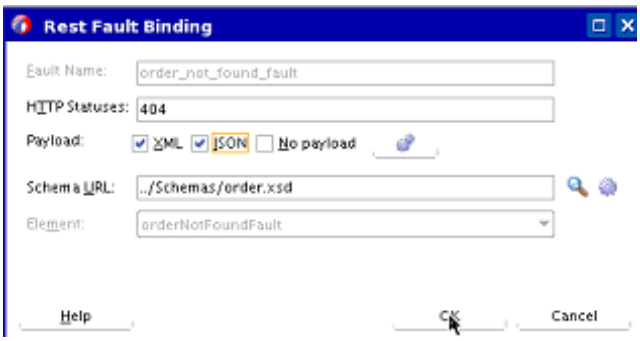
Table 36-4 Custom Header Support

Direction	Service Side	Reference Side
Request	Any HTTP headers that come in the request are propagated as normalized message properties. The headers are appended with **rest.binding.http.*header-name***. These headers are available in the service engine as part of normalized message properties. For example, any individual header can be obtained with the BPEL process in the receive and reply activity properties.	All normalized message properties prefixed with **rest.binding.http.*** are added as HTTP headers to the HTTP request. The REST service removes the prefix **rest.binding.http.** from the header name. before attaching the headers to the HTTP request.
Response	All normalized message properties prefixed with **rest.binding.http.*** are added as HTTP headers to the HTTP response. The REST service removes the prefix **rest.binding.http.** from the header name before attaching the headers to the HTTP response.	Any HTTP headers coming in the response are propagated as normalized message properties. The headers are appended with **rest.binding.http.*header-name***. These headers are available in the service engine as part of normalized message properties. For example, any individual header can be obtained with the BPEL process in the receive and reply activity properties.

What You May Need to Know About REST Fault Binding

You define REST fault binding response details in the REST Fault Binding dialog, as shown in Figure 36-24. If fault details are already defined in the WSDL file, a fault binding is automatically created in the **Fault Bindings** section of the REST Operation Binding dialog. You can also manually define fault bindings.by clicking the **Add fault binding** icon in the **Fault Bindings** section.

Figure 36-24 REST Fault Binding Dialog



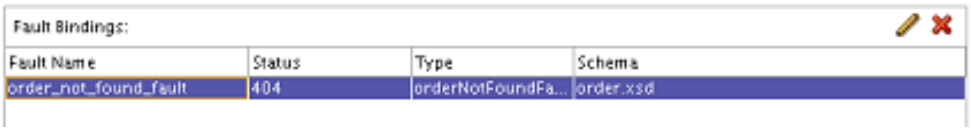
By default, the fault status is **400** when there is a fault payload and **404** when there is no fault payload.

Fault binding details are based on your selection in the Create REST Binding dialog:

- If you selected **REST enable component or reference**, fault bindings are automatically generated based on faults defined in the WSDL file.
- If you selected **Add operation binding**, you must configure the fault bindings to be supported, which are added to the WSDL being generated.

The **Fault Bindings** section of the REST Operation Binding dialogs shows the response fault name, type, status, and schema. Figure 36-25 provides details.

Figure 36-25 Fault Bindings Definition



What You May Need to Know About Converting a JSON Interchange Format to a REST Schema

You can select to create a REST schema from a JSON interchange format sample in the Choose Type dialog of the Native Format Builder wizard, as shown in Figure 36-22. During schema generation, the wizard attempts to do the following:

- Generate a REST schema with no namespace information
- Consume a JSON interchange format sample with no namespace information and generate an XML with the correct namespaces

However, there are several cases in which the conversion cannot be handled.

- Namespace information is retained to enable the JSON interchange format sample shown in [[LinkToOldExample - TopicID=SOASE88875 - Title=Unsupported JSON Interchange Format]] to be converted. This is because the underlying schema has elements and attributes from multiple namespaces.

  [[OldExample - - TopicID=SOASE88875 - Title=Unsupported JSON Interchange Format]]

```
<schema xmlns:us="http://xmlns.oracle.com/addresses/us"
xmlns:india="http://xmlns.oracle.com/addresses/india"
targetNamespace="http://xmlns.oracle.com"
xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
```

```
   <import ...>
   <element name="Person">
       <complexType>
           <choice>
                <element ref="us:Address"/>
                <element ref="india:Address"/>
           </choice>
       </complexType>
   </element>
</schema>
<schema targetNamespace="http://xmlns.oracle.com/addresses/us"
xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
   <element name="Address">
       <complexType>
           <sequence>
                <element name="Street" type="xsd:string"/>
                <element name="City" type="xsd:string"/>
                <element name="State" type="xsd:string"/>
                <element name="ZipCode" type="xsd:integer" minOccurs="0"/>
           </sequence>
       </complexType>
   </element>
</schema>
<schema targetNamespace="http://xmlns.oracle.com/addresses/india"
xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
   <element name="Address">
       <complexType>
           <sequence>
                <element name="Street" type="xsd:string"/>
                <element name="City" type="xsd:string"/>
                <element name="District" type="xsd:string" minOccurs="0"/>
                <element name="State" type="xsd:string"/>
                <element name="PinCode" type="xsd:integer" minOccurs="0"/>
           </sequence>
       </complexType>
   </element>
</schema>
```

- Sibling elements with duplicate names under a sequence group element cannot be converted because this translates to an object with duplicate keys in JSON, which is not valid.

What You May Need to Know About REST References Calling REST Services in the Same Node

The **SOAIncomingRequests_maxThreads** property by default is configured based in the **SOADataSource** data source in Oracle WebLogic Server Administration Console. This setting may be not enough for REST services under a heavy load (for example, if you have 200 concurrent users in a scenario in which a REST reference is calling a REST service within the same node). You must increase the **SOAIncomingRequests_maxThreads** value to `400` to avoid the exception error shown in [[LinkToOldExample - TopicID=SOASE88877 - Title=Exception Error When REST References Call REST Services]].

[[OldExample - - TopicID=SOASE88877 - Title=Exception Error When REST References Call REST Services]]

```
<May 2, 2014 10:16:11 AM PDT> <Error> <oracle.soa.bpel.system> <BEA-000000>
<cube engineJTA transaction is not in active state.
```

```
The transaction became inactive when executing activity "" for instance
"30,023", bpel engine can not proceed further without an active transaction.
please debug the invoked subsystem on why the transaction is not in active
status. the transaction status is "MARKED_ROLLBACK".
The reason was The execution of this instance "30023" for process
"BuyCoffeeBPELProcess" is supposed to be in an active jta transaction, the
current transaction status is "MARKED_ROLLBACK", the underlying exception is
"Service Unavailable" .
Consult the system administrator regarding this error.
, Cikey=30023, FlowId=20014, Current Activity Key=30023-BpInv0-BpSeq0.3-3,
Current Activity Label=InvokeCreateOrder,
ComponentDN=default/CoffeeShopClient!1.0*soa_19d4a881-115b-42c5-824d-1af3fa766
62d/BuyCoffeeBPELProcess
oracle.fabric.common.FabricInvocationException: Service Unavailable
        at
. . .
. . .
```

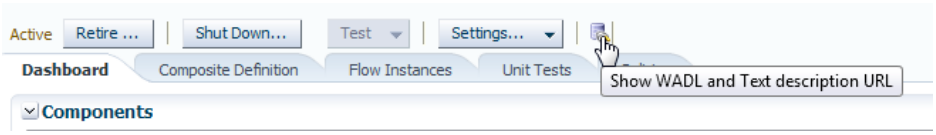
Testing the REST Adapter with the HTTP Analyzer

You can test the REST adapter with the HTTP Analyzer.

To test the REST adapter with the HTTP Analyzer:

1. Copy the WADL file URL from the home page of the SOA composite application in Oracle Enterprise Manager Fusion Middleware Control, as shown in Figure 36-26.

   Figure 36-26 WADL File URL in Oracle Enterprise Manager Fusion Middleware Control

2. In the HTTP Analyzer, click the **Open URL** icon, enter the WADL URL copied from Oracle Enterprise Manager Fusion Middleware Control, and press **Return**.

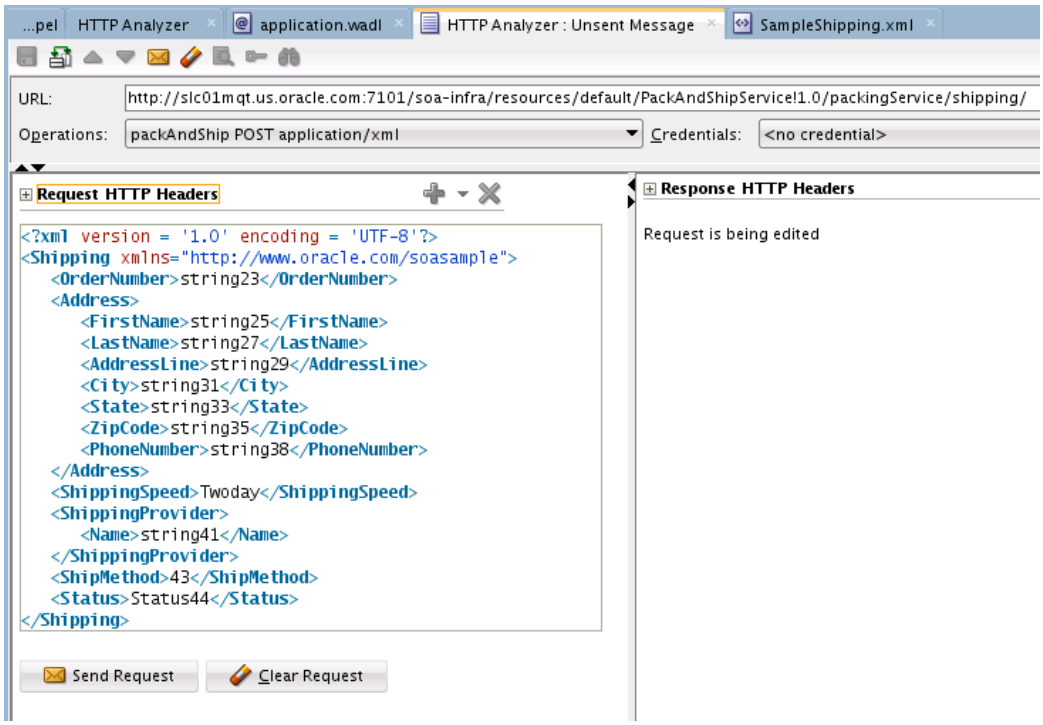
   The WADL file is included with the POST method. Figure 36-27 provides details.

Figure 36-27 WADL File and POST Method

3. Click **Test**.

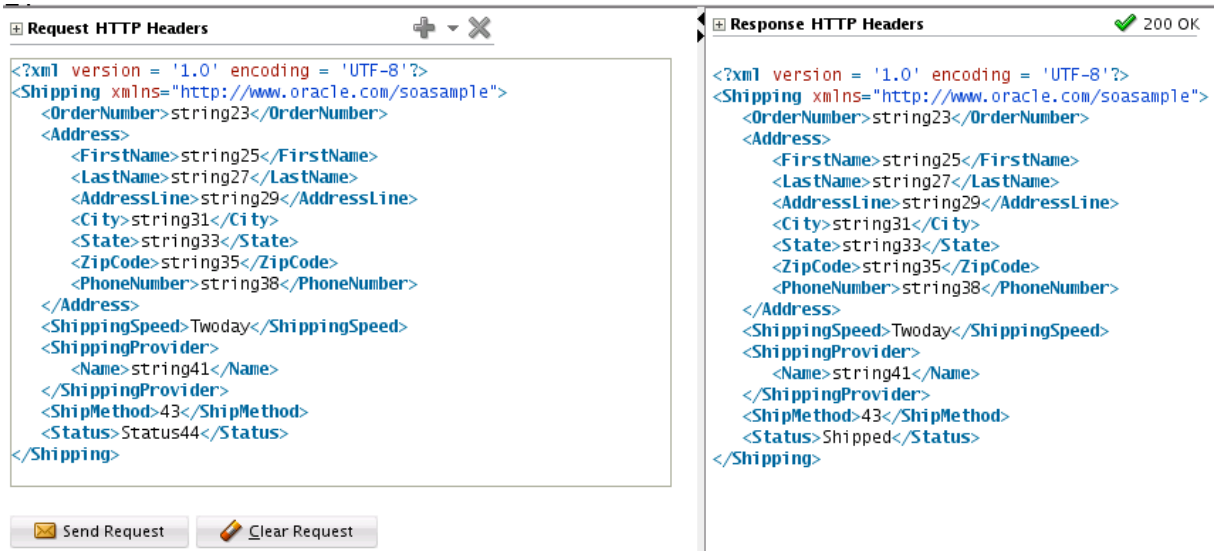
4. Copy and paste a sample request XML payload into the **Request HTTP Headers** section and click **Send Request**. You can also specify JSON formats. Figure 36-28 provides details.

Figure 36-28 Request Message



After processing completes, a response message is displayed. For this example, a message with an order status of **Shipped** is displayed. Figure 36-29 provides details.

Figure 36-29 Response Message



Testing and Configuring REST Reference Binding Components in Oracle Enterprise Manager Fusion Middleware Control

You can initiate instances of SOA composite applications that include REST binding components from the Test Instances page in Oracle Enterprise Manager Fusion Middleware Control. This page enables you to test any WSDL or WADL. For more information, see Section "Initiating a SOA Composite Application Test Instance" of *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

You can configure properties for REST reference binding components in Oracle Enterprise Manager Fusion Middleware Control. For more information, see Section "Configuring Properties for REST Adapters" of *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

37

Integrating Enterprise JavaBeans with Composite Applications

This chapter describes how to integrate Enterprise JavaBeans with SOA composite applications through use of Java interfaces or service data object (SDO) parameters. It describes how to design an SDO-based Enterprise JavaBeans application, create an Enterprise JavaBeans service in Oracle JDeveloper, design an Enterprise JavaBeans client to invoke Oracle SOA Suite, specify Enterprise JavaBeans roles, and configure JNDI access.

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).

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 Java interfaces (does not use a WSDL file to define the interface)
- Through use of SDO-based EJBs (uses 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 Integrating the Spring Framework in SOA Composite Applications.

Integration Through Java Interfaces

You can 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 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 the example that follows. The Java interface classes must be compatible with the WSDL file used by the connecting components (that is, if a message is sent to a BPEL component). BPEL services are defined with a WSDL, and the Java interface classes must be compatible with that WSDL.

```
<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 *Solutions Guide for Oracle TopLink* and Integrating the Spring Framework in SOA Composite Applications.

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 37-1 provides an overview.

Figure 37-1 SDO and Enterprise JavaBeans Binding Integration

You use the Create EJB Service dialog in Oracle JDeveloper to define this integration, as described in 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 the following example:

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

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:

- *Developing Enterprise JavaBeans for Oracle WebLogic Server*
- *Developing Fusion Web Applications with Oracle Application Development Framework*
- Oracle JDeveloper online help table of contents for the following topics:
  - Enterprise JavaBeans
  - SDO for Enterprise JavaBeans/Java Persistence API (JPA)

  Access the help by selecting **Help** > **Table of Contents** in Oracle JDeveloper.

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 WSDL Chooser dialog, which is accessible from the **WSDL URL** field of the Create EJB Service dialog (described in Creating an Enterprise JavaBeans Service in).

  – 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).

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.

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.

How to Define the SDO Types with an Enterprise JavaBeans Bean

An Enterprise JavaBeans bean must define the SDO types. The example that follows 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.

```
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 entries shown in the following example:

```
<?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 37-2 provides a code example of a session bean with SDO logic defined.

Figure 37-2 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);
          }
```

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 37-1 describes the annotations to use.

Table 37-1 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.

Table 37-1 (Cont.) Annotations

Name	Description
`@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.
`@javax.xml.ws.RequestWrapper;` `@javax.xml.ws.ResponseWrapper;`	Tells the Enterprise JavaBeans binding components whether the deserialized object must be unwrapped or whether a wrapper must be created before serialization. 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 service endpoint interface (SEI) class name is used instead.

Add appropriate method parameter annotations	Controls 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")`

The following example provides an example of an 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);
}
```

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.

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 a Java interface
- Integration through an SDO interface

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

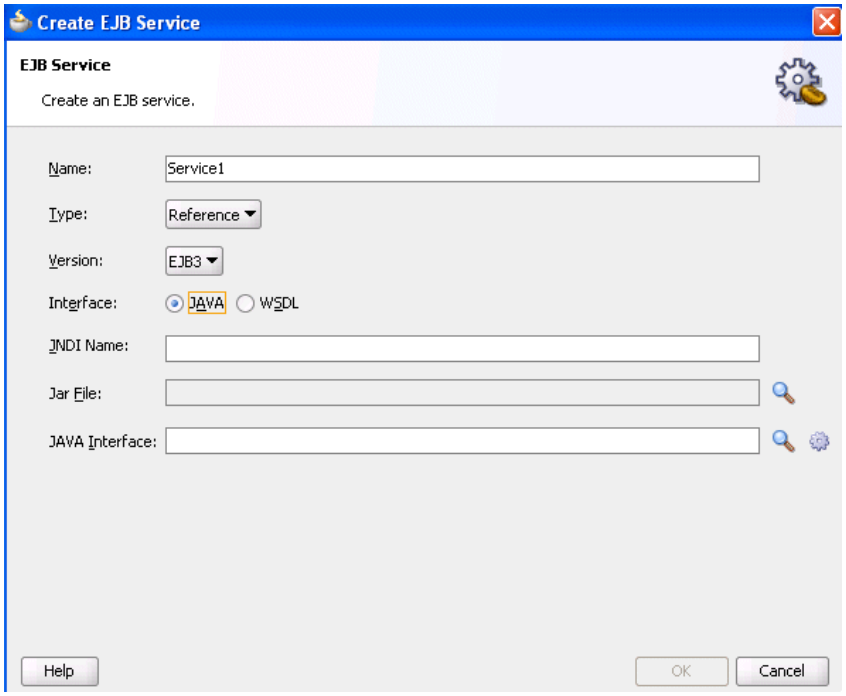
To integrate Java interface-based Enterprise JavaBeans with SOA composite applications:

1. Go to the SOA composite application in the SOA Composite Editor.
2. In the **Technology** section of the Components window, drag the **EJB** 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.

3. In the **Interface** section, click **Java** (if it is not already selected).
4. The Create EJB Service dialog displays the fields shown in Figure 37-3.

Figure 37-3 Create EJB Service for Java Interface



5. Enter the details shown in Table 37-2. The fields are the same regardless of the swimlane in which you dragged the **EJB** icon.

Table 37-2 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.

Table 37-2 (Cont.) Create EJB Service Dialog

Field	Value
Jar File	Click the **Search** icon to select the EJB JAR file created in Designing an SDO-Based Enterprise JavaBeans Application. The JAR Chooser 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 are 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 How to Integrate SDO-based Enterprise JavaBeans with SOA Composite Applications.

6. Click **OK**.

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. Go to the SOA composite application in the SOA Composite Editor.
2. In the **Technology** section of the Components window, drag the **EJB** icon into the appropriate swimlane, as described in Table 37-3.

Table 37-3 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**

The Create EJB Service dialog is displayed.

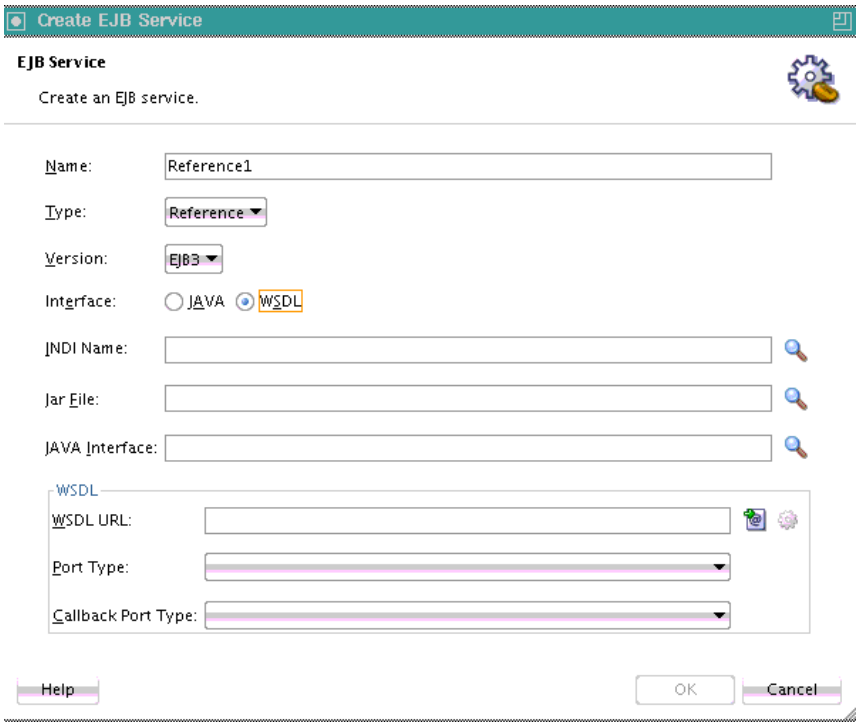
3. In the **Interface** section, click **WSDL**.
4. See the step in Table 37-4 based on the swimlane in which you dragged the **EJB** service.

Table 37-4 Swimlane Location

If You Dragged the EJB Service to this Swimlane...	Then Go To...
External References	44.a
Exposed Services	44.b

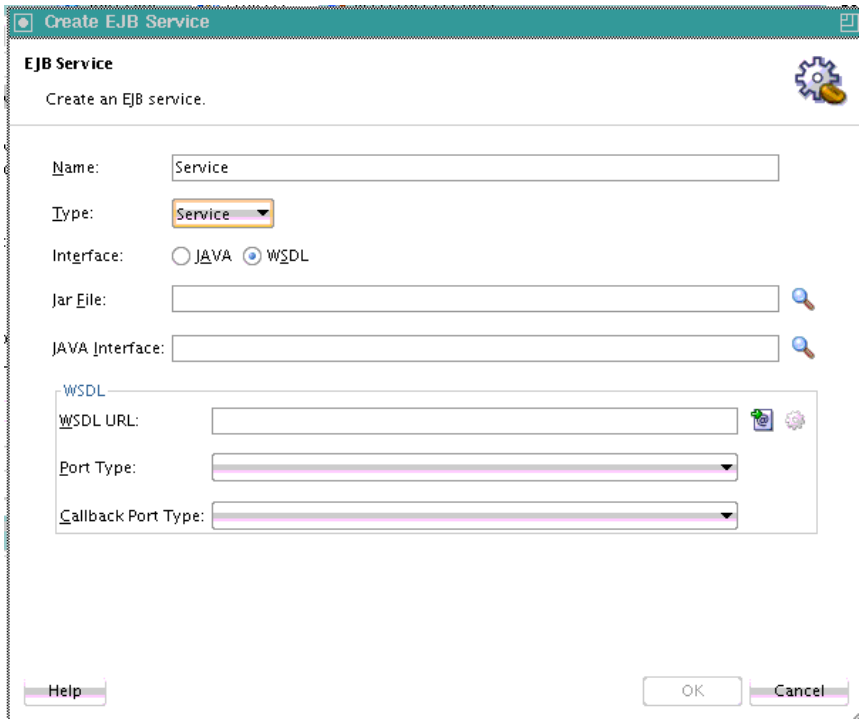
a. View the Create EJB Service dialog that displays in the **External References** swimlane, as shown in Figure 37-4.

Figure 37-4 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 37-5.

Figure 37-5 Create EJB Service in Exposed Services Swimlane



5. Enter values appropriate to your environment. The fields that display differ based on the swimlane in which you dragged the **EJB Service** icon. Table 37-5 provides details.

Table 37-5 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	**Note**: This field only displays if you dragged the **EJB Service** icon into the **External References** swimlane. Select the version of EJB to support: **EJB2** or **EJB3** (the default selection). If you select **WSDL** from the **Interface** list, only **EJB3** is available for selection.
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.

Table 37-5 (Cont.) Create EJB Service Dialog

Field	Value
Jar File	Click the **Search** icon to select the EJB JAR file created in Designing an SDO-Based Enterprise JavaBeans Application. The JAR Chooser 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 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 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).

6. Click **OK**.

Designing an Enterprise JavaBeans Client to Invoke Oracle SOA Suite

This section describes how to design an Enterprise JavaBeans client to invoke Oracle SOA Suite.

How to Create a Java Interface-Based Client to Invoke Oracle SOA Suite

Use the standard Enterprise JavaBeans client. The following example provides details:

```
InitialContext ic = new InitialContext(jndiProps);
SimpleEjb svc = (SimpleEjb) ic.lookup("PassthroughRef");
String result = svc.addBreadCrumb("RemoteTest");
```

How to Invoke an SDO-Enterprise JavaBeans Service

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 (`$FMW_HOME/soa/soa/modules/oracle.soa.fabric_11.1.1/fabric-client.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. The code that follows provides an example.

  You must specify the complete path of the service ID with the `MyTestEJBService` parameter of `serviceFactory.createService` (for example, `"default/MyTestProject!1.0/MyTestEJBService"`). If the complete path is not specified, you receive an `EJBException- Could not locate the service` error.

```
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("complete_path/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();
```

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. The following example provides details:

```
<service name="EJBService" ui:wsdlLocation="BPELEJBProcess.wsdl">
    <interface.wsdl
interface="http://xmlns.oracle.com/EJBApplication/EJBProject/BPELEJBProcess#wsdl.int
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>
```

Configuring Enterprise JavaBeans Binding Support in the Credential Store Framework

This section describes how to configure Enterprise JavaBeans binding support in the credential store framework.

How to Configure Enterprise JavaBeans Binding Support in the Credential Store Framework

All Enterprise JavaBeans bindings support using the Credential Store Framework (CSF) to store JNDI user access credentials, and not just service data object (SDO) Enterprise JavaBeans bindings.

You can edit the following Enterprise JavaBeans binding JNDI properties in Oracle Enterprise Manager Fusion Middleware Control:

- `java.naming.factory.initial`
- `java.naming.provider.url`
- `java.naming.dns.url`
- `java.naming.factory.url.pkgs`
- `java.naming.factory.url.pkgs`
- `java.naming.security.authentication`
- `java.naming.security.protocol`
- `java.naming.security.principal`
- `java.naming.security.crendentials`
- `oracle.jps.credstore.map`
- `oracle.jps.credstore.key`

To configure Enterprise JavaBeans binding support in the credential store framework:

To edit these properties, perform the following steps in Oracle Enterprise Manager Fusion Middleware Control:

1. Right-click the SOA composite application that includes the Enterprise JavaBeans binding component.
2. Select **Service/Reference Properties**.
3. Select the Enterprise JavaBeans binding component.
4. Click the **Properties** tab.
5. Set the appropriate properties.

To specify the oracle.jps.credstore.map and oracle.jps.credstore.key properties

Oracle recommends that you store the JNDI lookup principal/credentials in the CSF map by specifying the properties `oracle.jps.credstore.map` and `oracle.jps.credstore.key`. Storing the user name/password directly as properties is not secure.

1. In Oracle Enterprise Manager Fusion Middleware Control, navigate to one of the following to display the Credentials page.
   a. **Domain** > **Security** > **Credentials** (if the application is deployed on Oracle WebLogic Server).

      or

   b. **Cell** > **Security** > **Application Policies** (if it is deployed on WebSphere Application Server).
2. To add a new map, select **Create Map**.
3. Click the map to add a key entry for `oracle.jps.credstore.map`.
4. Repeat Steps 2 and 3 to add `oracle.jps.credstore.key`.

To grant SOA infrastructure runtime access to the CSF map store

After completing these steps, you must grant SOA Infrastructure runtime access to the CSF map store.

1. Expand the **WebLogic Domain**.
2. Right-click **soa-infra**, and select **Security** > **System Policies**.
3. Search for type **CodeBase**, which includes the name **fabric-runtime**.
4. Select the entry and edit it to add a credential store access permission.
5. Grant at least the read action to the map.

38

Using Direct Binding to Invoke Composite Services

This chapter describes the Direct Binding Invocation API and how to invoke a SOA composite application. It describes how to create an inbound direct binding service, how to create an outbound direct binding reference, and how to set an identity for Java 2 Platform, Standard Edition (J2SE) clients invoking direct binding. Samples of using the Direct Binding Invocation API are also provided.

This chapter includes the following sections:

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 a remote method invocation (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 the following example:

```
<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 (see `oracle.soa.management.facade.Locator`).

- `ConnectionFactory` class name (see `oracle.soa.management.facade.Locator`).

  The `ConnectionFactory` class must implement the `oracle.soa.api.invocation.DirectConnectFactory` interface.

  If the `ConnectionFactory` class name is not specified, the default `oracle.soa.api.JNDIDirectConnectionFactory` is used. To use the default connection factory, you must supply the lookup name for the EJB.

- 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 a secure socket layer (SSL) for the callback JNDI connection. If this flag is set to `true`, then the WS-Addressing `replyTo` header instructs the service to call back at an SSL JNDI port.

A sample configuration is shown in the following example:

```
<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 38-1 describes a sample synchronous invocation pattern and Figure 38-2 describes a sample asynchronous invocation pattern.

Figure 38-1 Sample Synchronous Invocation Patterns

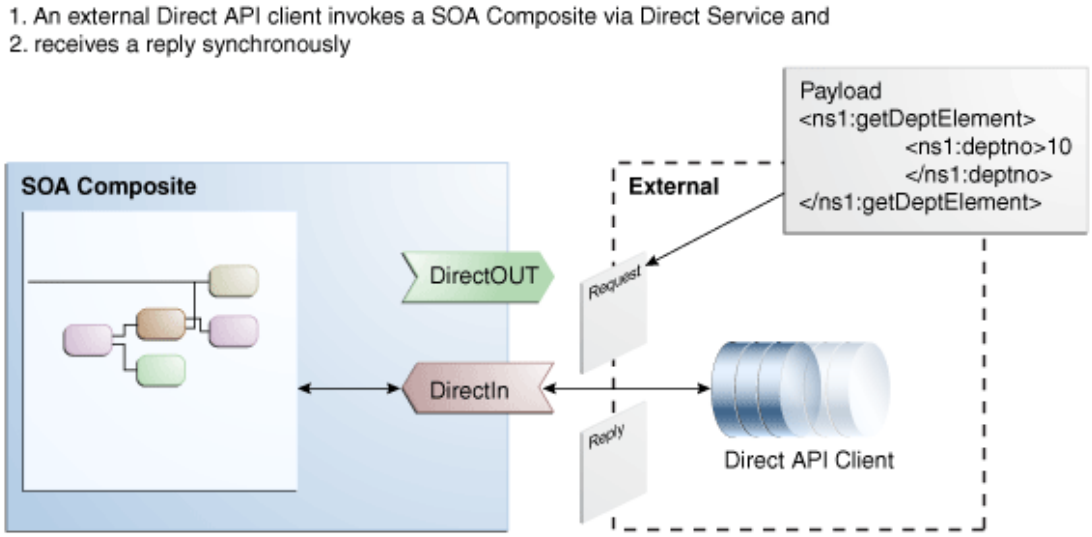
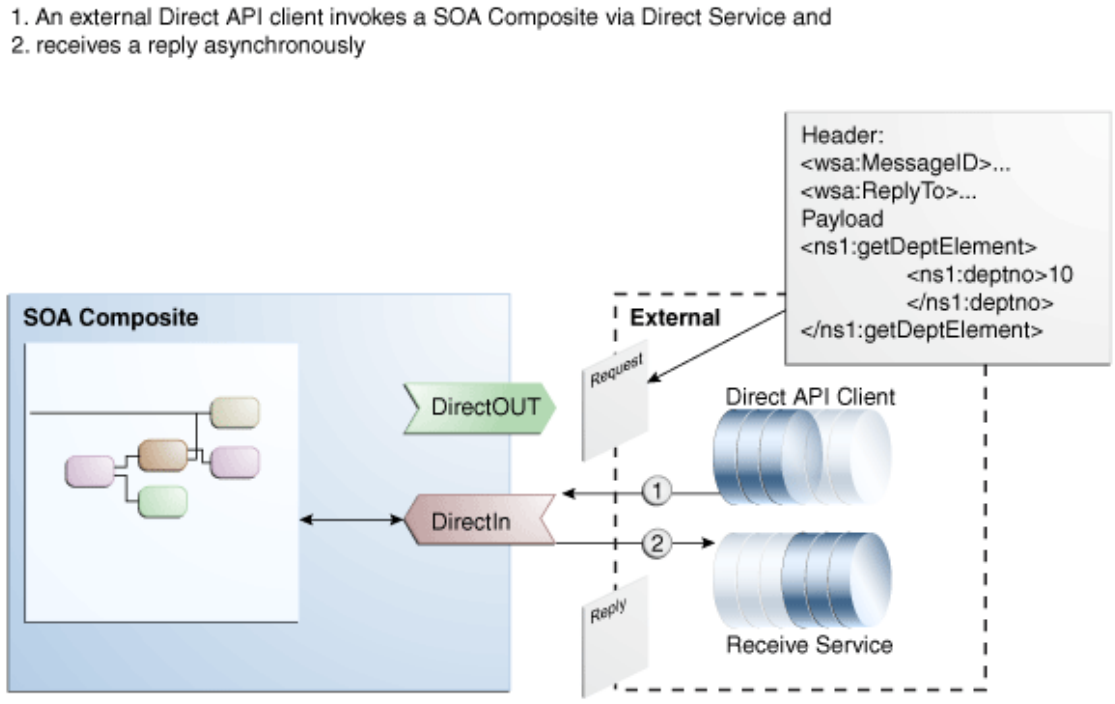


Figure 38-2 Sample Asynchronous Invocation Pattern



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 the following example for returning the `DirectConnection`.

```
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 the following example:

```
Hashtable jndiProps = new Hashtable();
jndiProps.put(Context.PROVIDER_URL, "t3://" + hostname + ':' + portname + "/soa-
infra");
jndiProps.put(Context.INITIAL_CONTEXT_FACTORY,"weblogic.jndi.WLInitialContextFacto
ry");
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 *Java API Reference for Infrastructure Management*.

- `oracle.soa.api.message.Message`

  The `Message` interface encapsulates the data exchanged. For more information, see *Java API Reference for Infrastructure Management*.

Synchronous Direct Binding Invocation

Direct binding also supports the synchronous direct invocation with use of the method shown in the following example:

```
<T> Message<T> request(String operationName, Message<T> message)
 throws InvocationException, FaultException
```

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 below:

```
<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:

- `replyToAddress` = `sb://testserver:9001/callback`
- `replyToReferenceParameter`: `element of WSA:ReferenceParameters`

Required JAR Files for Compiling and Running the Direct Binding Java Client Code

The following JAR file is required for compiling the direct binding Java client code:

- `$FMWHOME/soa/soa/modules/oracle.soa.mgmt_11.1.1/soa-infra-mgmt.jar`

The following JAR files are required for running the direct binding Java client code:

- `$FMWHOME/wlserver/server/lib/wlthint3client.jar`
- `$FMWHOME/soa/soa/modules/oracle.soa.fabric_11.1.1/oracle-soa-client-api.jar`

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 (`label` is optional):

```
domainName/compositeName[!compositeVersion[*label]]
```

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 during any operation in the SOA composite, then roll back the transaction from the client side. This rolls back all the database operations.

Invoking a SOA Composite Application in Oracle JDeveloper with the Invocation API

The **Direct** icon in the Components window in Oracle JDeveloper, as shown in Figure 38-3, provides support for exchanging SOA messages with SOA over RMI.

Figure 38-3 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`, `wlthint3client.jar`, and `oracle-soa-client-api.jar`.

For more information about the Direct Binding Invocation API, see Introduction to the Direct Binding Invocation API.

How to Create an Inbound Direct Binding Service

You can invoke a SOA composite application using the **Direct** icon in the Components window in Oracle JDeveloper.

To create an inbound direct binding service:

1. Open Oracle JDeveloper.
2. From the Components window, select **SOA**.
3. From the **Technology** list, drag the **Direct** icon into the **Exposed Services** swimlane. The Create Direct Binding dialog appears.
4. Enter the details shown in Table 38-1.

Table 38-1 Create Direct Binding Dialog Fields and Values

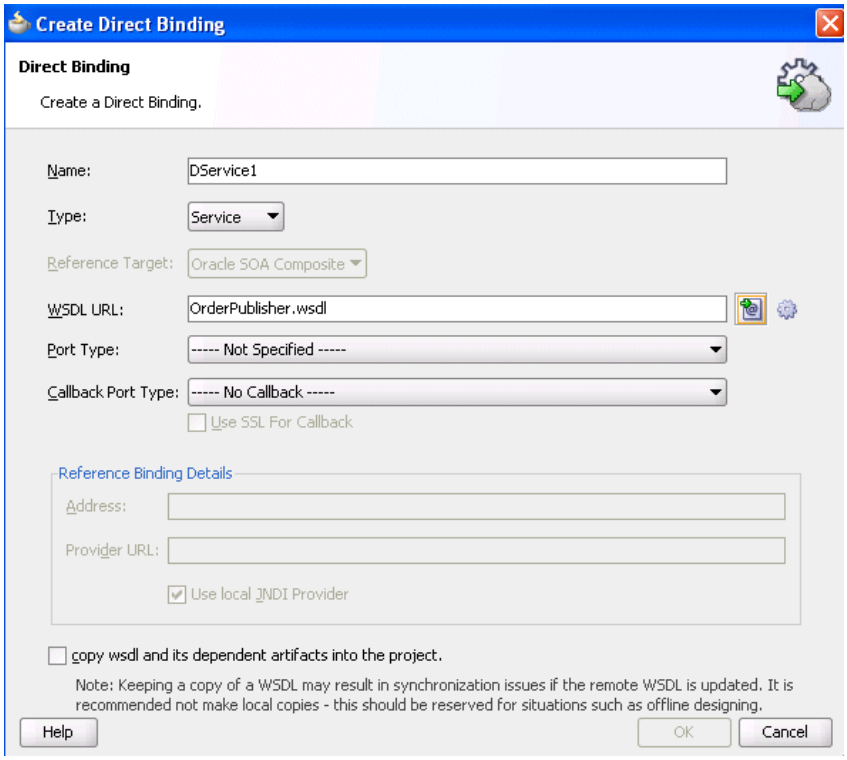
Field	Value
Name	Enter a name.
Type	Select **Service** from the list.
Reference Target	This field is disabled when defining this service in the **Exposed Services** swimlane.
WSDL URL	The URL location of the WSDL file. If you have an existing WSDL, then click the **Find Existing WSDLs** option. Otherwise, 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.

Table 38-1 (Cont.) Create Direct Binding Dialog Fields and Values

Field	Value
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 check box. If you select this check box, 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 38-4.

Figure 38-4 Create Direct Binding Dialog



5. Click **OK**.

The direct binding service displays in the SOA Composite Editor shown in Figure 38-5. The single arrow in a circle indicates that this is a synchronous, one-way, direct binding component.

Figure 38-5 Direct Binding Service

How to Create an Outbound Direct Binding Reference

You can create an outbound direct binding reference using the **Direct** icon in the Components window in Oracle JDeveloper to either invoke a SOA composite application or an Oracle Service Bus.

Note:

When Oracle SOA Suite and Oracle Service Bus are in different domains, you must enable trust between the domains.

To create an outbound direct binding reference:

1. Open Oracle JDeveloper.
2. From the Components window, select **SOA**.
3. From the **Technology** list, drag the **Direct** icon into the **External References** swimlane. The Create Direct Binding dialog appears.
4. Enter the details shown in Table 38-2.

Table 38-2 Create Direct Binding Dialog Fields and Values

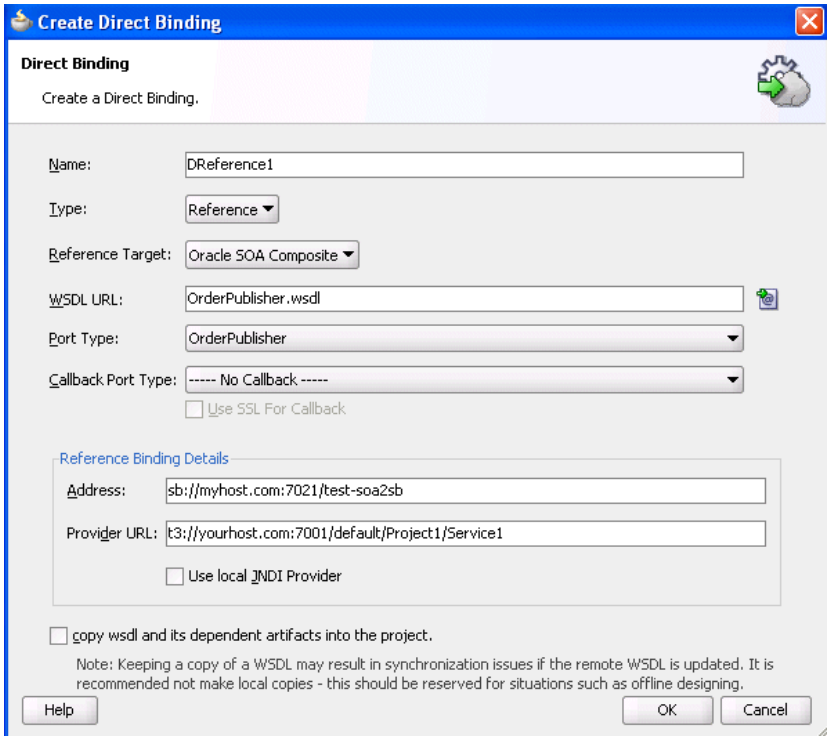
Field	Value
Name	Enter a name.
Type	Select **Reference** from the list.
Reference Target	Select the reference target on which you want the direct binding service to operate: • **Oracle SOA Composite**: Creates a direct binding with a SOA composite application as a reference target. • **Oracle Service 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.

Table 38-2 (Cont.) Create Direct Binding Dialog Fields and Values

Field	Value
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 check box. If you select this check box, 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 38-6. For more information about using the Oracle SOA Suite services with Oracle Service Bus, see Chapter "Oracle SOA Suite Transport (SOA-DIRECT)" of *Developing Services with Oracle Service Bus*.

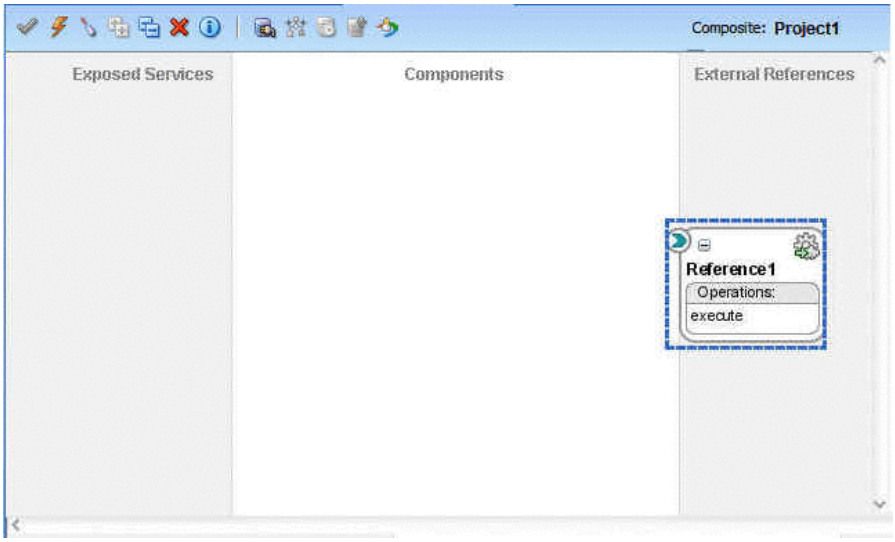
Figure 38-6 Create Direct Binding Dialog



5. Click **OK**.

The direct binding reference displays in the designer shown in Figure 38-7. The single arrow in a circle indicates that this is a synchronous, one-way direct binding reference component.

Figure 38-7 Direct Binding Reference



How to Set an Identity for J2SE Clients Invoking Direct Binding

A user identity can be established when authenticating to the server during the process of JNDI lookup by passing the JNDI security credential, as shown in the following example:

```
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();
    }
}
```

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.

Samples Using the Direct Binding Invocation API

This section provides some examples of how the API is used. It 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.

```
// 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);
properties.put(Context.SECURITY_CREDENTIALS, PASSWORD);
DirectConnection conn =
    dcFactory.createConnection("soadirect:/default/MyComposite!1.0/MyService",
                               properties);

// 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);

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:

39

Oracle SOA Suite Templates and Reusable Subprocesses

This chapter describes how to create and use Oracle SOA Suite templates in SOA projects, service components, and BPEL scope activities and how to create and reuse standalone and inline BPEL subprocesses within other processes.

This chapter includes the following sections:

Introduction to Oracle SOA Suite Templates

A template is a reusable part of an Oracle SOA Suite project that you can use to create new projects. There are three types of templates, as described in Table 39-1.

Table 39-1 Template Types

Template Type	Description
SOA project	A complete SOA project packaged and used to start new projects. You can create new SOA composite applications using this template.
Service component	A service component, such as a BPEL 2.0 process (including sensors) packaged for import into other projects. All dependent components and wires are also packaged. It appears as a custom service component in the SOA composite application's Components window.
Custom BPEL scope activity	A scope activity of a BPEL process packaged as a custom activity in the Components window and ready for import into other BPEL projects. This custom activity can potentially surface in the BPEL activity palette of the Components window.

Oracle SOA Suite templates provide the following benefits:

- Share common code (subpart of a process or a scope) between applications, composites, and processes. You create once, then share with others. The template can be reused multiple times.

- Store and reuse templates from the Oracle Metadata Services Repository (MDS Repository).
- Fully editable upon consumption.
- Automatically discover templates in Oracle JDeveloper. Once the template is saved, it is displayed in the Components window.
- No inheritance, meaning that future changes to source templates are not visible to applications. If you make changes to the source template, a current user of the template does not see the change.
- Custom icons are provided for component scope templates.
- No versioning in templates. To differentiate between versions, you specify the version number in the template name.
- Support for templates in both the BPEL versions 1.1 and 2.0.

Changes made to a specific template are not propagated to projects previously created using this template. This functionality is achievable through layered customization.

A new annotation is added to the composites/BPEL processes to identify the relationship to a template.

For information about using templates, see Creating and Using a SOA Project Template, Creating and Using a Service Component Template, and Creating and Using a BPEL Scope Activity Template.

Introduction to Standalone and Inline BPEL Subprocess Invocations

BPEL provides limited support for modularizing business process logic for reusability. The only method is to package reusable process logic as completely separate processes, which are utilized by the parent process (the process utilizing the reusable process logic) in a method identical to using a web service (through the invoke activity).

To address this challenge, Oracle SOA Suite provides a subprocess extension to BPEL. A subprocess is a fragment of BPEL code that can be reused within a particular processor by separate processes. The subprocess extension provides the following benefits:

- BPEL process code reusability, which reduces the need to create the same activities multiple times to perform the same tasks.
- Code modularity.
- Code maintenance (changes are propagated, which eliminates the need to implement updates in multiple places every time a change is necessary).
- Less overhead than invoke activities.
- Memory footprint reduction, which can be considerable in a complex process.

Note:

- Subprocesses are only supported with BPEL version 2.0. There is no support with BPEL version 1.1.
- Correlation sets are not supported in subprocesses. If you create a correlation set in an inline or standalone subprocess, it fails during runtime.
- Subprocesses cannot be shared between multiple composites.
- Monitor view is not supported from inside a subprocess. Monitor view is accessible from a BPEL process by selecting the **Change to Monitor view** icon above Oracle BPEL Designer.

Oracle SOA Suite provides support for two types of subprocesses, as described in Table 39-2.

Table 39-2 Subprocess Types

Standalone Subprocess	Inline Subprocess
• A BPEL call activity invokes the subprocess. • Supports subprocesses in the same composite only. • Visible in the Components window. • Does not have an interface and can only be called from another BPEL process. It can include partner links. • A fragment of a BPEL process that includes a number of activities that are reused across other BPEL processes. • In the composite view, the wire to a subprocess is shown as a dotted line to indicate that this is not a wire between actual components.	• A BPEL call activity invokes the subprocess. • Part of the parent BPEL process code and not visible in the composite view. • Visible in the Components window. • Subprocess code is re-entrant and reusable at runtime:. Only one copy is stored in memory, even if called many times. • For groups of activities that are reused within *one* BPEL process. • Can either define parameters to set or can use the process parameters.
Not Applicable.	• Activities must be in a scope activity to be converted into a subprocess.
For information about creating a standalone subprocess, see How to Create a Standalone BPEL Subprocess.	For information about creating an inline subprocess, see How to Create an Inline Subprocess.

Introduction to a Standalone Subprocess

A standalone subprocess is defined, as shown in the following example, in a file with the extension `.sbpel` (subprocess BPEL extension).

```
<!-- A subprocess is defined in a SBPEL file, containing a bpelx:subProcess
 ! document
 ! The bpelx:subProcess is similar to a standard bpel:process, with
```

```
 ! differences asnoted below.
-->

<bpelx:subProcess name="NCName" targetNamespace="anyURI"
   xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
   xmlns:bpelx="http://schemas.oracle.com/bpel/extension" ...>

   <!-- Partner links and variables serve as sub-process arguments -->
   <partnerLinks>?
      <partnerLink name="NCName" partnerLinkType="QName" myRole="NCName"?
         partnerRole="NCName"?
         bpelx:argumentRequired=["yes"|"no"]? />
   <partnerLinks>
   <variables>?
      <variable name="BPELVariableName"  messageType="QName"? type="QName"?
         element="QName"?
         bpelx:argumentRequired=["yes"|"no"]?>
         from-spec?
      </variable>
   </variables>

  <!-- Standard process definition here, except no <receive> or <pick> with -->
  <!--  createInstance="yes" -->
  /activity/
</bpelx:subProcess>
```

The `<subProcess>` element is an extension of the WS-BPEL 2.0 language. The `<subProcess>` element is the root element for a subprocess definition. The namespace for this element is as follows:

```
http://schemas.oracle.com/bpel/extension
```

The `<subProcess>` activity must be embedded in an `<extensionActivity>`, as specified in section 10.9 of the .

A subprocess is of type `tProcess`, as defined in the following WS-BPEL target namespace:

```
http://docs.oasis-open.org/wsbpel/2.0/process/executable
```

It differs from `tProcess` in the following ways:

- Variables and partner links immediately under the `<subProcess>` element can serve as arguments for the subprocess. Required arguments are marked by setting the attribute `argumentRequired` to `yes` (the default value is `no`). The subprocess's required arguments are the minimum set of arguments the caller must pass to it.
- A variable defined with an inline `from-spec` initializer serves as an optional argument with a default value. If the caller passes this argument, the caller-supplied value for the argument overrides the default value.
- Validation reports an error if a variable is referenced prior to setting the value if it is not a required argument.
- The first activity in the subprocess cannot be a receive or pick activity with `createInstance` set to `yes`. This is because no instance of a given subprocess type is created; the subprocess is logically part of an existing process instance.

The subprocess `/@name` attribute defines the name of the subprocess that is unique within the composite in which it is deployed.

The subprocess is self-contained. That is, all the variable and partner link references in the process snippet resolve to local definitions or arguments. This contrasts with the `<inlineSubProcess>` element, which allows unresolved references to variables and partner links that are in-scope at the call activity.

In a typical scenario, more than one variable is exchanged between the parent and a subprocess. If they are large documents, copying them is expensive. Because of this, passing by reference is an option.

A subprocess can converse with partners synchronously (`InOut`) or asynchronously (`InOnly`). The partner link for these interactions can be passed as an argument from a parent process or configured within the subprocess. For asynchronous requests, the conversation ID for WS-Addressing/normalized messages is set with the parent process instance ID. This enables routing of callback messages to the correct process instance.

Subprocesses in a SOA composite application are enumerated in the `composite.xml` file. The component element definition associates a subprocess's name with the `sbpel` file in which it is defined. During deployment, the subprocess components are delegated to the BPEL process service engine. The BPEL process service engine validates the process definition and builds a map with the subprocess target name as the key and the subprocess definition as the value. At most, only one instance of a subprocess exists in the service engine independent of consumer count. For optimizing memory, it may lazily load the process or unload the process if it is not actively used.

For information about creating a standalone subprocess, see How to Create a Standalone BPEL Subprocess.

Introduction to an Inline Subprocess

An inline subprocess can be defined as part of a BPEL 2.0 process at the `<process>` level. The syntax is shown in the following example:

```
<process name="NCName" targetNamespace="anyURI"
         xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable" ...>
    <!--
     ! All sub-process definitions must appear prior to the WS-BPEL artifacts of
     ! the process definition.
    -->

    <!-- Inline sub-process definition at process scope -->
    <bpelx:inlineSubProcess xmlns:bpelx="http://schemas.oracle.com/bpel/extension"
      name="NCName">*
      ...
      <!-- Partner links and variables serve as sub-process arguments -->
      <partnerLinks>?
        <partnerLink name="NCName" partnerLinkType="QName" myRole="NCName"?
          partnerRole="NCName"?
      bpelx:argumentRequired=["yes"|"no"]? />+
      <partnerLinks>
      ...
      <variables>?
        <variable name="BPELVariableName"  messageType="QName"? type="QName"?
          element="QName"?
                  bpelx:argumentRequired=["yes"|"no"]?>+
          from-spec?
        </variable>
      </variables>
      ...
      <!--
```

```
       ! Standard process activity graph here, except that no <receive> or <pick>
       ! activities with createInstance = "yes" are allowed.
      -->
      activity
    </bpelx:inlineSubprocess>

    <!--
     ! BPEL code stripped for brevity
    -->
</process>
```

When a BPEL process instance is first created, all subprocess references are resolved. When the process executes a particular call activity, it uses the subprocess resolved at instance creation time. Therefore, two different instances of the same process may use different versions of subprocesses referenced if, for example, the default composite revision for a subprocess changes.

When the BPEL process instance executes the call activity, it is executed within the process's execution space, sharing its state. The call activity transfers control to the subprocess, at which time the subprocess scope is initialized with the argument variables:

- Each parameter is copied (by reference or value, as specified) from the call activity to the subprocess's scope.
- Optional parameters (those with default values) that are not referred to in the call activity's parameter list are initialized with their default values.
- All required parameters must be supplied by the call activity.
- All values supplied by the call activity's parameters must be type-compatible with the corresponding variable (or partner link) defined in the subprocess.
- Each variable (or partner link) in the subprocess can be set only once in a call activity's parameter list.

On completion of the subprocess, control is returned to the parent process. In the normal case, execution continues with the next activity after the call activity. In the case of abnormal subprocess completion, the parent process evolves the process according to the standard life cycle rules of WS-BPEL.

From the monitoring and management view, there is no new process instance for the subprocess created. It is represented by a call activity in the parent process instance. Expanding the activity (navigate) shows subprocess execution details.

To minimize linking errors during runtime, upon deployment of the process and subprocess, references are resolved. Parameter lists are validated as a postdeployment activity. Preprocessing for creating a new process instance validates all subprocess references in the process. If any reference is not resolved, the instance is *not* created. Instead, an error message is returned, meaning essentially the following:

```
HTTP Status Code 503, "service not available
```

Upon a linking error, if the service consumer is waiting, an error message is sent to the consumer that is inline with exit activity handling. Otherwise, the instance is suspended with the reason set as `linkage error`. If a suitable subprocess is deployed and the reference is resolvable, suspended instances can then be recovered and resume normal execution by automatic recovery.

For information about creating an inline subprocess, see How to Create an Inline Subprocess.

Differences Between Oracle SOA Suite Templates and Reusable Subprocesses

When determining whether templates or reusable subprocesses are the best solution for your business use case, it is important to understand the differences:

- Templates

  A template is a customizable, skeletal project, service component, or scope activity. You can drag and drop a template into a SOA composite application or a BPEL process and make additional changes. You essentially are copying and pasting a template. For example, if there are 50 lines of code in a template and you copy it twice to use, the code increases by 100 lines.

- Reusable subprocesses

  A subprocess is a BPEL code snippet intended for a specific purpose. A subprocess that is defined earlier can be called and used as it is. An inline subprocess of 50 lines can be called twice and the parent process code remains at 50 lines, and not 100. Subprocesses perform better and have a smaller memory foot print than templates.

Creating Oracle SOA Suite Templates

You can create the following types of templates:

- SOA project
- Service component
- Custom BPEL scope activity

For conceptual information about templates, see Introduction to Templates and Differences Between Oracle SOA Suite Templates and Reusable Subprocesses.

Creating and Using a SOA Project Template

This section describes how to create and use a SOA project as a template.

Note:

Use of templates is not supported in the Oracle JDeveloper Customization role.

How To Create a SOA Project Template

To create a SOA project template:

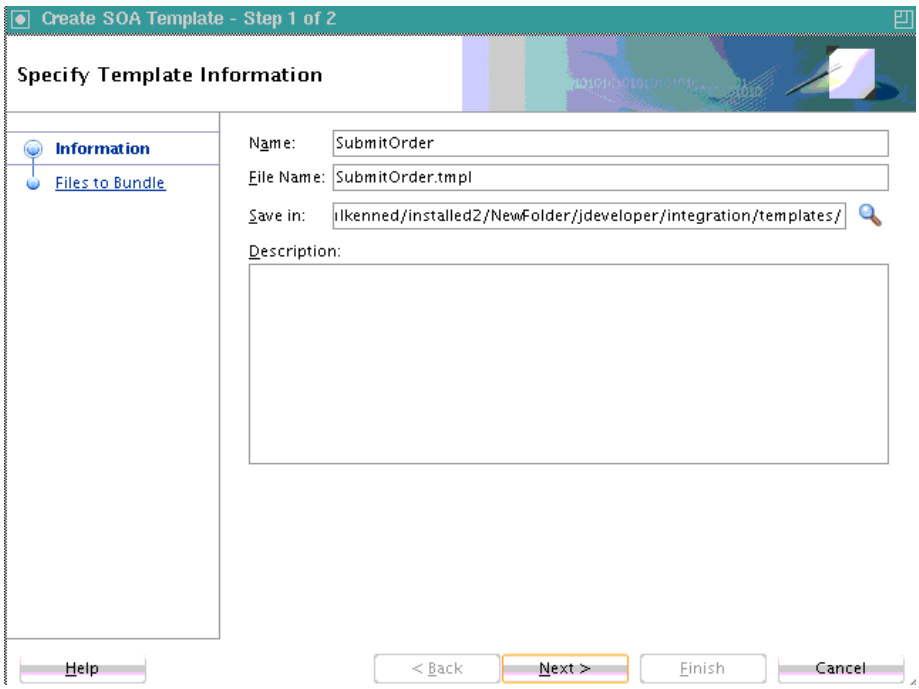
1. Open a SOA composite application.
2. In the Applications window, right-click either of the following:
   - The ***composite_name***

- The project name

3. Select **Create SOA Template**.

   This invokes the Create SOA Template wizard. Default names and the location for saving the template based on the composite name are automatically included. Figure 39-1 provides details.

Figure 39-1 Create SOA Template Wizard - Specify Template Information Page



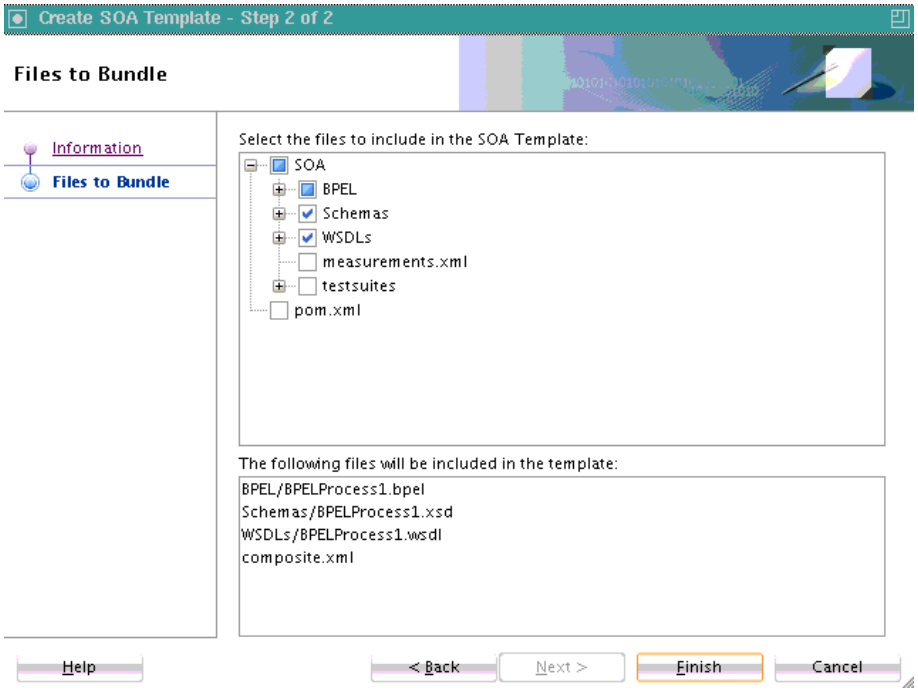
4. Change the default values and enter a description, as necessary, and click **Next**. The **Browse** icon for the **Save in** field enables you to save the template in the file system or the Oracle SOA Suite design time section of the MDS Repository.

   The Create SOA Template Wizard - Files to Bundle page is displayed. Figure 39-2 provides details. This page shows all the files packaged as part of this template.

   You can also manually select measurements (business indicators) and test suites to include. If your composite includes domain value maps (DVMs) (for example, a DVM function is referenced in a BPEL scope activity), they are also included in the template.

   For information about business indicators, see Configuring BPEL Process Analytics. For information about test suites, see Introduction to the Composite Test Framework.

Figure 39-2 Create SOA Template Wizard - Files to Bundle Page



5. View the files to package and select additional files, and click **Finish**.
6. Click **OK** when prompted to acknowledge that the template was successfully created.

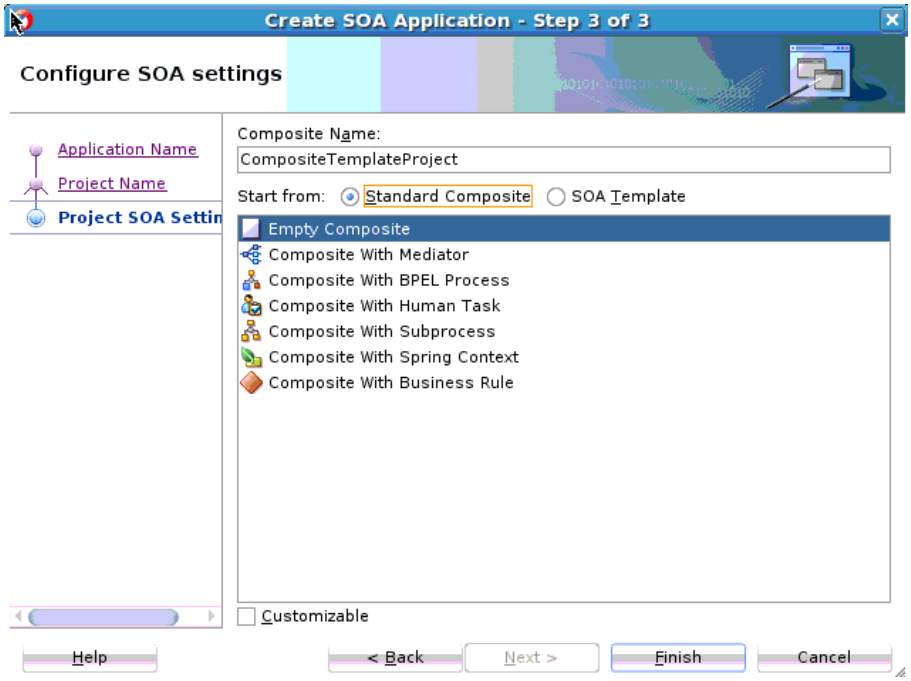
How to Use a Composite Template in Another SOA Composite

This section describes how to use the composite template created in How To Create a SOA Project Template in another SOA composite application.

To use a composite template in another SOA composite

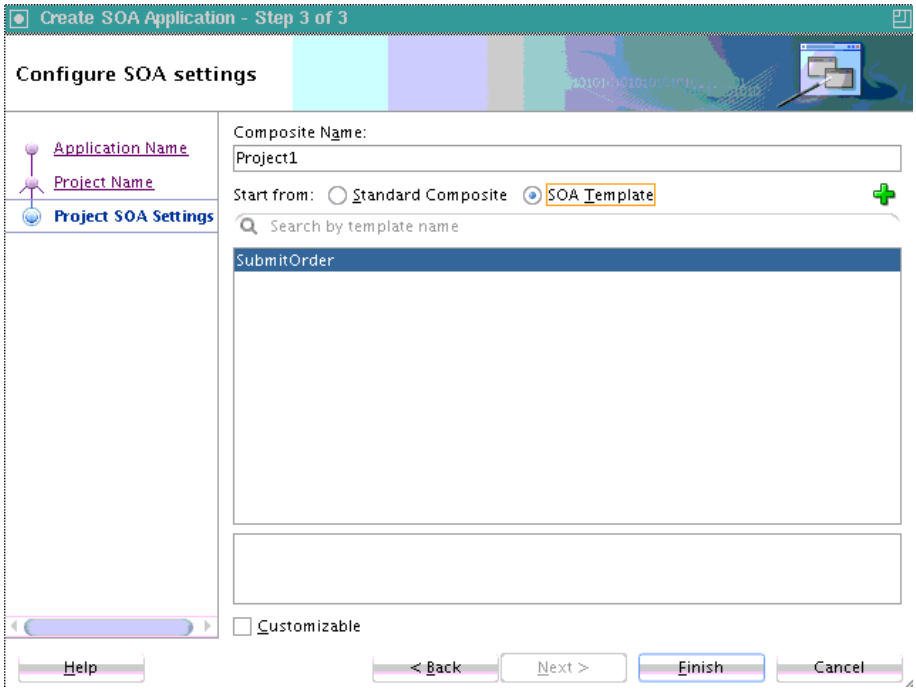
1. Create a new SOA composite application in Oracle JDeveloper.
2. On the Create SOA Application wizard - Configure SOA Settings page, select **SOA Template**. Figure 39-3 provides details.

Figure 39-3 Custom Template Selection



The list of available templates is displayed. Figure 39-4 provides details.

Figure 39-4 SOA Templates Available for Selection



3. Select a template from the list, or click **Add** to select additional templates.
4. Click **Finish**.

   The SOA Composite Editor is displayed with the custom template. The files of the template are displayed in the Applications window.

You can rename components as necessary, such as renaming the binding components and process names.

5. Right-click and select **Rename**.

Creating and Using a Service Component Template

This section describes how to create and use a service component template.

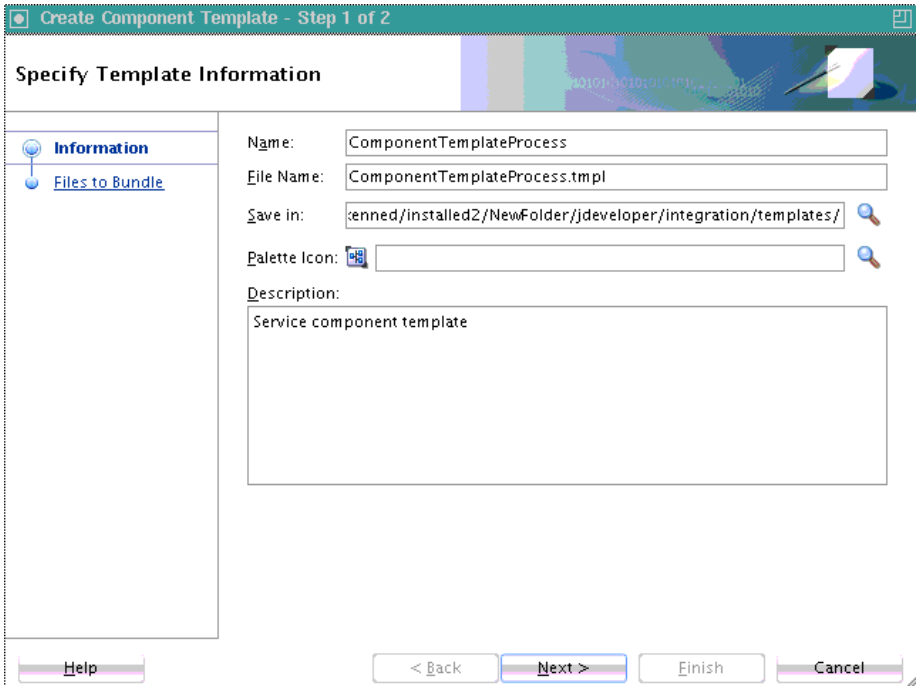
How to Create a Service Component Template

To create a service component template:

1. From the Oracle JDeveloper main menu, select **File** > **New**.
2. Select **SOA Project**, and click **OK**.
3. Enter a project name, and click **Next**.
4. Select a BPEL project, and click **Finish**.
5. Design a SOA composite application.
6. In the SOA Composite Editor, right-click the service component from which to create a template.
7. Select **Create Component Template**.

   This launches the Create Component Template wizard.
8. Provide appropriate responses, including optionally selecting an icon for the partner link, and click **Next**. Figure 39-5 provides details.

Figure 39-5 Create Component Template Wizard - Specify Template Information Page



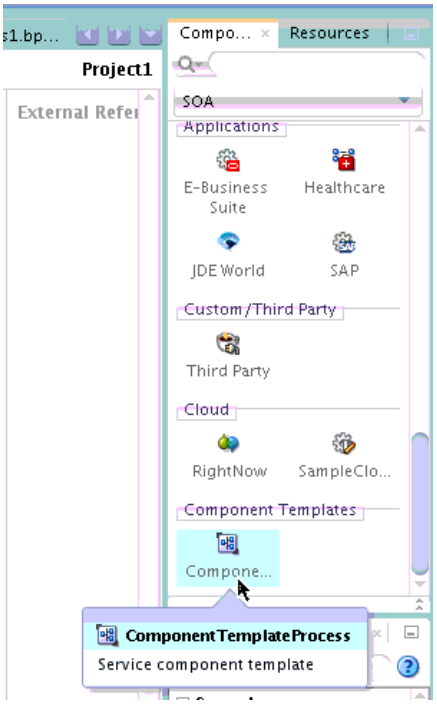
The Create Component Template wizard - Files to Bundle Page is displayed.

9. View the files packaged and select additional files (such as adapters and measurements), as required, and click **Finish**.

10. Click **OK** when prompted to acknowledge that the template was successfully created.

    The service component template is added to the **Component Templates** section of the Components window. Figure 39-6 provides details.

Figure 39-6 Service Component Template in Component Templates Section of Components Window



How to Use a Service Component Template in Another SOA Composite

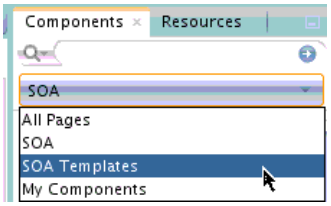
This section describes how to use the packaged service component template created in How to Create a Service Component Template in another SOA composite application.

To use a service component template in another SOA composite:

1. Create an empty SOA composite application in Oracle JDeveloper.

2. In the SOA Composite Editor, select **SOA Templates** from the **SOA** list. Figure 39-7 provides details.

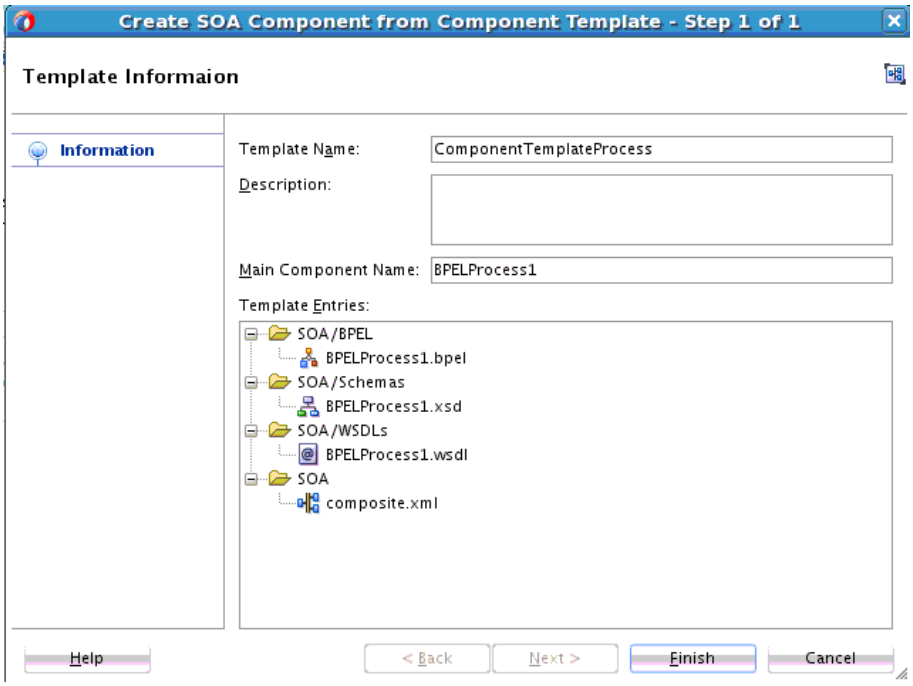
Figure 39-7 SOA Templates Option in SOA Menu



3. Drag the service component template into the SOA Composite Editor.

   This invokes the Create SOA Component from Component Template dialog, as shown in Figure 39-8. This dialog shows the template name, description, and files included in the template.

Figure 39-8 Create SOA Component from Component Template Dialog



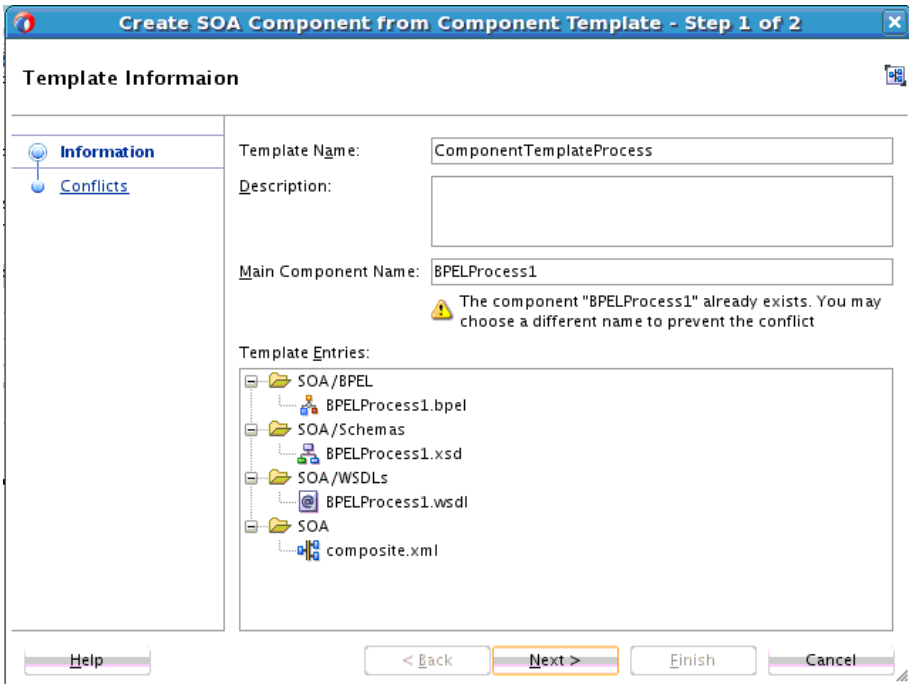
4. Click **OK**.

   The service component template is displayed in the SOA composite application.

5. View the Applications window and note that files such as schemas and WSDLs are displayed in the SOA composite application.

6. If you attempt to apply the service component template a second time to the same SOA composite application, the Create SOA Component from Component Template dialog is displayed and indicates that there is a conflict because schema and BPEL files are already in the composite. Figure 39-9 provides details.

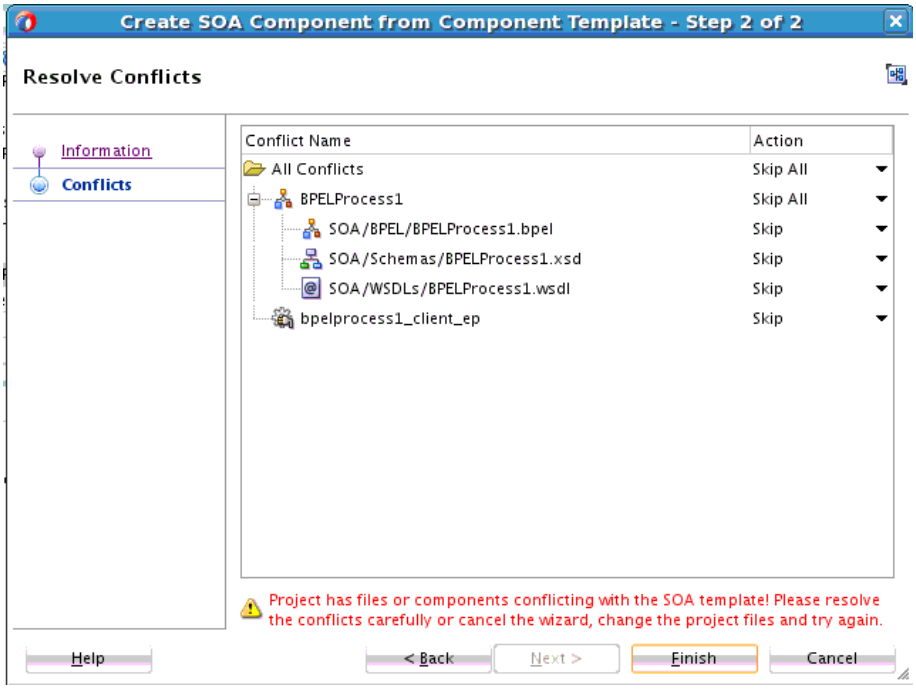
Figure 39-9 Create SOA Component from Component Template Dialog



7. Click **Next**.

8. In the Resolve Conflicts page, select to skip or overwrite all files or specific files that are in conflict. Figure 39-10 provides details.

Figure 39-10 File Names in Conflict



9. When complete, click **Finish**.

Creating and Using a BPEL Scope Activity Template

This section describes how to create and use a BPEL scope activity template.

How to Create a BPEL Scope Activity Template

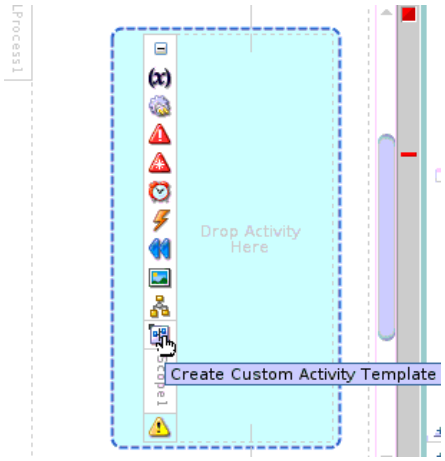
To create a BPEL scope activity template:

1. In Oracle BPEL Designer, drag a scope activity into a BPEL process.
2. Design the contents of the scope activity to include activities, event handlers, and catch and catch all branches that include fault variables, as necessary.
3. Create a template from the scope.
   a. Right-click the scope and select **Create Custom Activity Template**.

   or

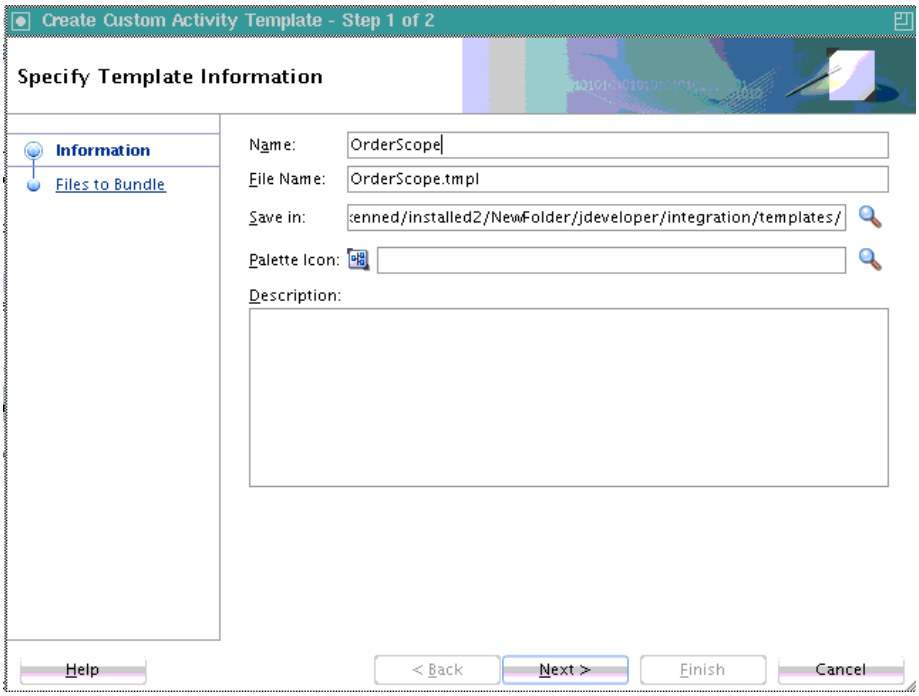
   a. Expand the scope and select **Create Custom Activity Template**, as shown in Figure 39-11.

Figure 39-11 Scope Template Creation



The Create Custom Activity Template wizard - Specify Template Information page is displayed, as shown in Figure 39-12.

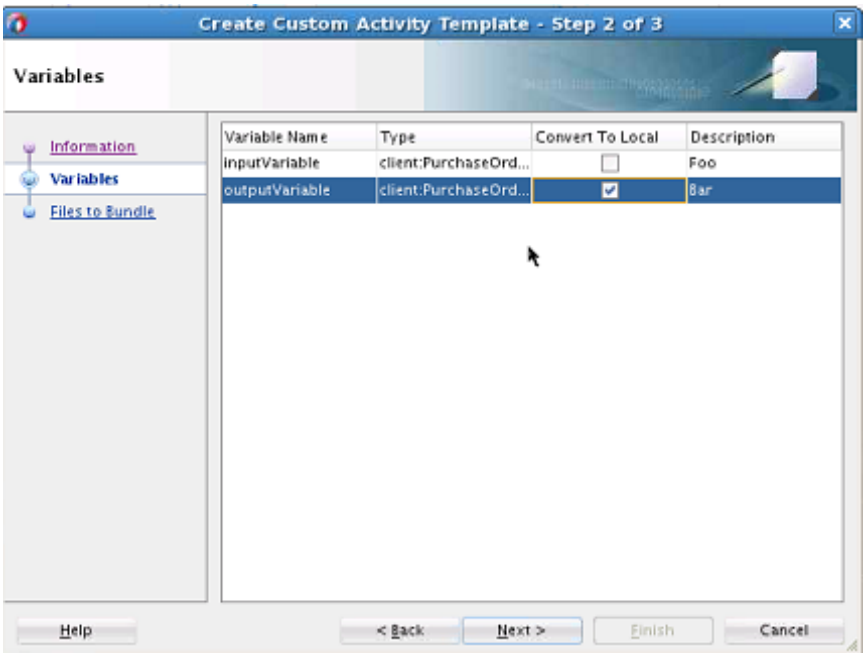
Figure 39-12 Create Custom Activity Template Wizard - Specify Template Information Page



4. Specify details, and click **Next**.

   The Create Custom Activity Template wizard - Variables page is displayed. Figure 39-13 provides details. This page is displayed if variables are used in the scope. This page is not displayed if you have an empty scope or a scope that does not use variables.

Figure 39-13 Create Custom Activity Template Wizard - Variable Page



5. Select to convert your variables to local variables. This conversion is not recommended if this variable is used outside of the scope activity in receive and reply activities. If the variables are used only inside this scope, the check boxes are selected by default.

6. Enter an optional description of the variables, and click **Next**.

   The Create Custom Activity Template wizard - Files to Bundle page is displayed as shown previously in Figure 39-2. This page shows all the files packaged as part of this template. You can also manually select test suites to include.

7. Select files, and click **Finish**.

How to Use a BPEL Scope Activity Template in Another BPEL Process

This section describes how to use a BPEL scope activity template in another BPEL process.

To use a BPEL scope activity template in another BPEL process

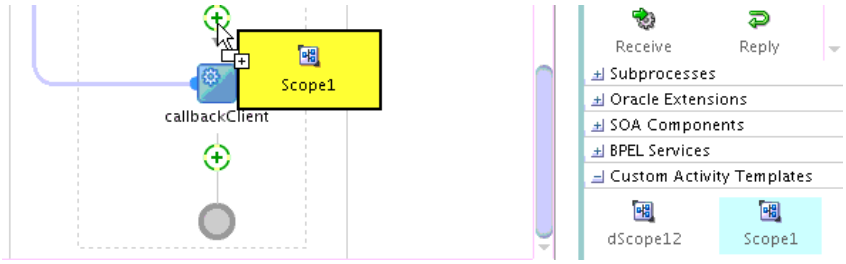
1. Create a new or open an existing BPEL process.

2. From the **Custom Activity Templates** section in the Components window, drag the scope activity template created in How to Create a BPEL Scope Activity Template into the BPEL process. Figure 39-14 provides details.

   Note:

   Only scope activity templates that are compatible with the BPEL service component version are available. For example, if this is a BPEL 2.0 service component, only scope activity templates for BPEL 2.0 are available for selection. No BPEL version 1.1 scope activity templates are displayed.

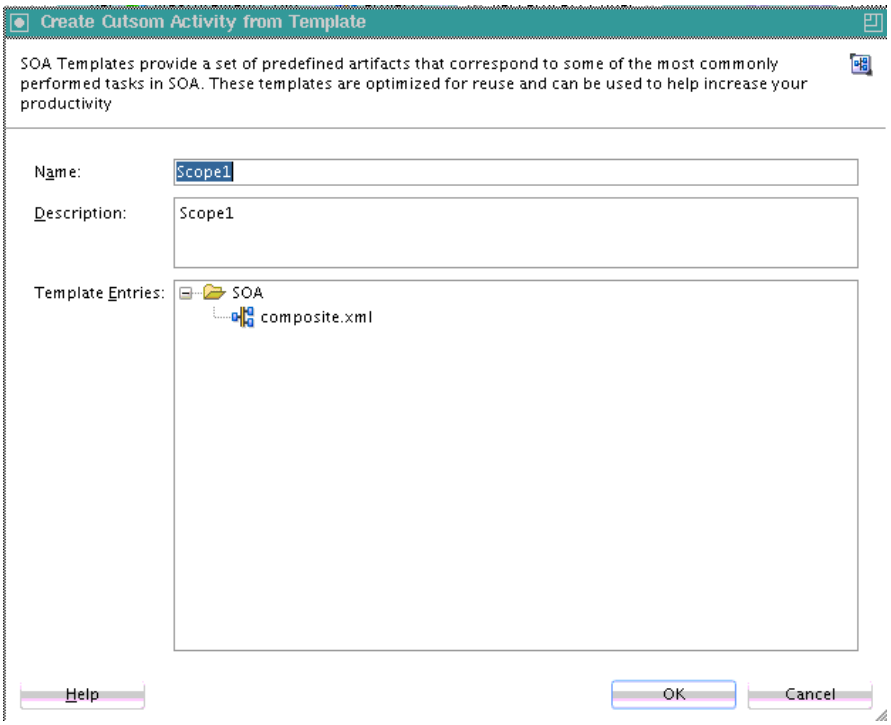
   Figure 39-14 Scope Activity Template

   Any error handling you designed such as catch and catch all activities and any scope variables you created are also copied into the BPEL process.

   The Create Custom Activity from Template page is displayed, as shown in Figure 39-15.

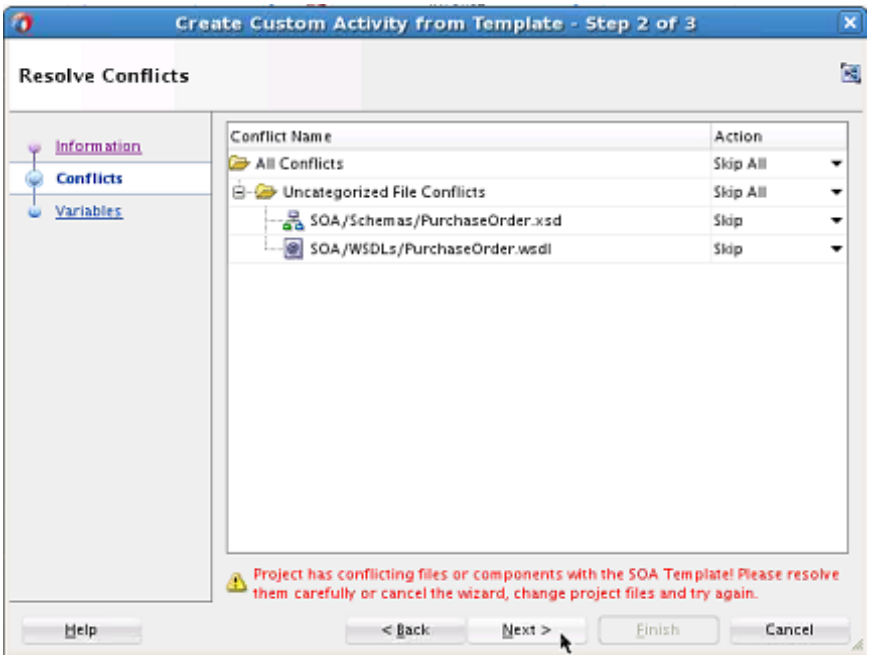
Figure 39-15 Create Custom Activity from Template Wizard



3. Click **Next**.

   If there are conflicts, the Create Custom Activity from Template wizard - Resolve Conflicts page is displayed, as shown in Figure 39-16.

Figure 39-16 Create Custom Activity from Template Wizard - Resolve Conflicts Page

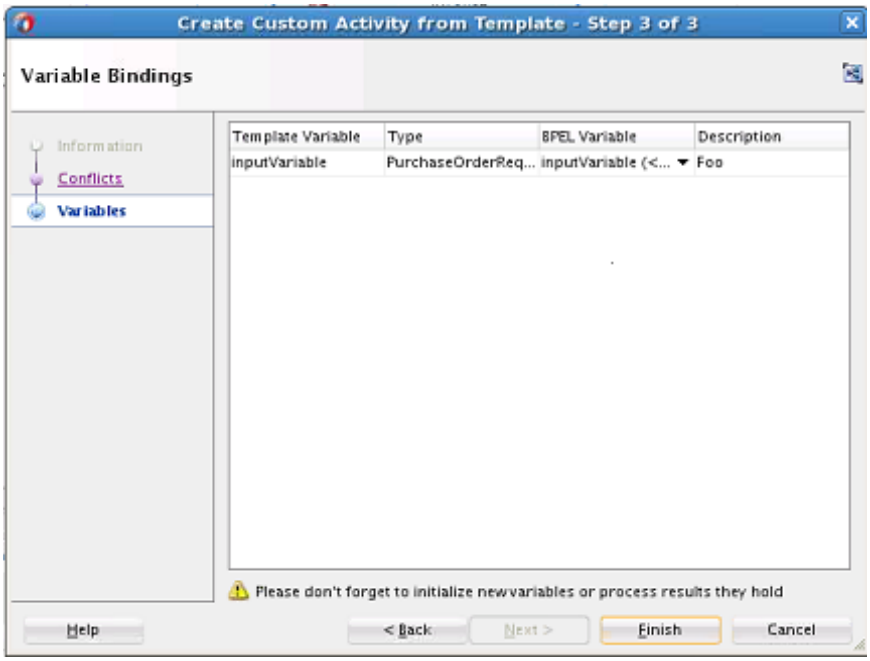


4. Select to skip all or individual file conflicts, and click **Next**.

The Create Custom Activity from Template wizard - Variables page is displayed, as shown in Figure 39-17.

If you selected to convert your variables to local variables on the Create Custom Activity Template Wizard - Variable Page in Step 5 of How to Create a BPEL Scope Activity Template, they do not require special processing and are not displayed on this page. Only variables that were not converted to local variables are displayed on the Create Custom Activity from Template wizard - Variable Bindings page.

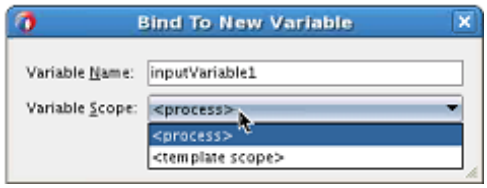
Figure 39-17 Create Custom Activity from Template wizard - Variable Bindings Page



5. If the template and the project both include this variable, you can choose to reuse the variable or bind to a new variable from the list in the **BPEL Variable** column.

   a. If you selected to bind to a new variable, enter a name and select whether to create the variable locally for the template scope or globally for the BPEL process, then click **OK**. Figure 39-18 provides details.

   Figure 39-18 Bind to New Variable Dialog

   If you drop an activity template inside of Scope A that is inside of Scope B, then Scope A and Scope B also are in the list. This enables you to select among all locations where variables can be declared.

6. Click **Next**.

7. If a scope uses partner links, the Create Custom Activity Template Wizard - Partner Links Page is displayed.

8. Click **Finish**.

Managing Templates

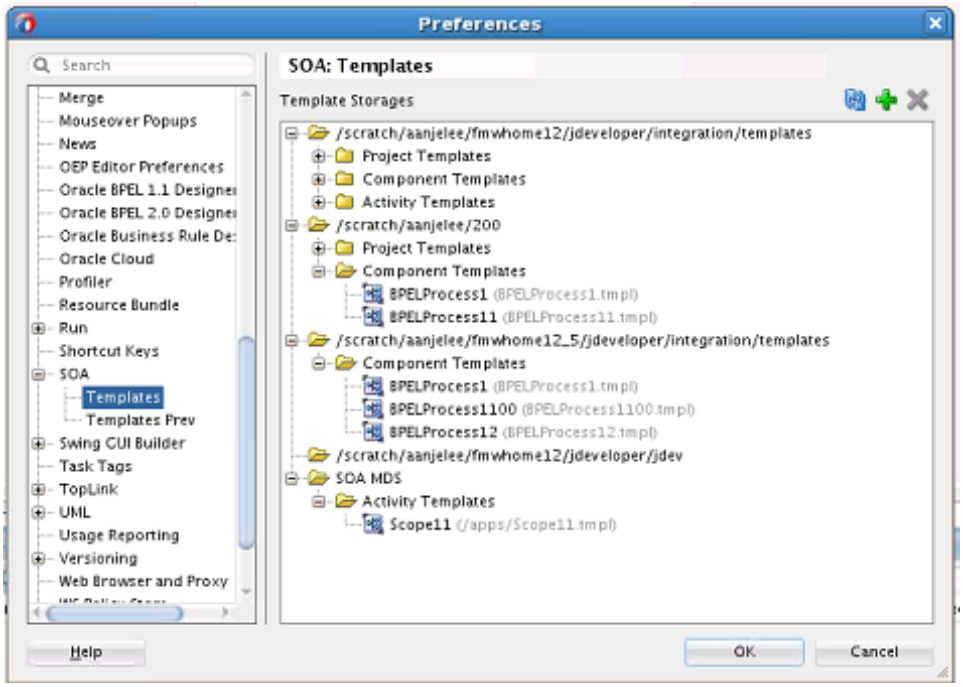
You can manage all available template types from the Preferences dialog.

To manage templates:

1. From the Oracle JDeveloper main menu, select **Tools** > **Preferences** > **SOA** > **Templates**.

   The Preference dialog is displayed, as shown in Figure 39-19.

   Figure 39-19 SOA Template Preferences



   Templates can be stored in two locations:

   - Folders: Templates are stored in the file system.
   - SOA-MDS: Templates are stored in the MDS Repository and can be shared.

2. Right-click a folder to display a list of management tasks, as shown in Figure 39-20.

Figure 39-20 Management Tasks

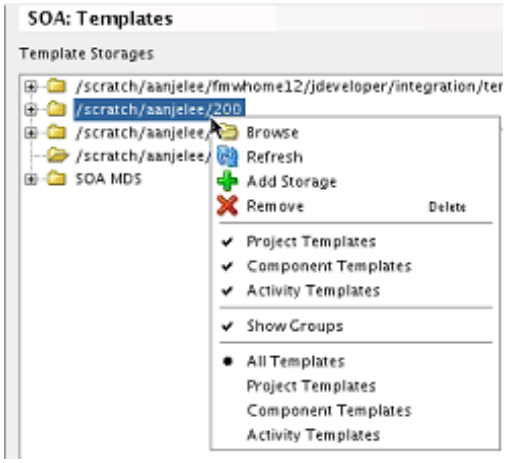


Table 39-3 describes the management tasks you can perform.

Table 39-3 Template Management Tasks

Element	Description
Browse	Browses for a specific template name. The **Browse** option uses Windows Explorer on Windows or the file browser on Linux for the storage folder. Templates are stored as files, so you may want to operate with them as with files (that is, upload with FTP, send by email, copy to another folder to back up, and so on).
Refresh	Refreshes the list of templates.
Add Storage	Adds existing templates to the Preferences - SOA Templates dialog.
Remove	Deletes the folder and its templates only from the Preferences - SOA Templates dialog. The templates are *not* physically deleted from the file system or MDS Repository. You can add them to this dialog again by selecting **Add Storage** or clicking the **Add** icon. The template context menu contains a **Delete** option that physically deletes a template.
All Templates	Displays all templates.
Project Templates	Displays only SOA project templates.
Component Templates	Displays only service component templates.
Activity Templates	Displays only BPEL scope activity templates.

3. If you want to import a template to the `jdeveloper/integration/templates` directory, select **File** > **Import** > **SOA Template**. The file can then be added to the Preferences dialog by clicking the **Add** icon or right-clicking a folder and selecting **Add Storage**.

Creating Standalone and Inline BPEL Subprocesses in a BPEL Process

You can create standalone subprocesses in a SOA composite and inline BPEL subprocesses in a BPEL process. A subprocess is a fragment of BPEL code that can be reused within a particular processor by separate processes.

For conceptual information about subprocesses, see Introduction to Standalone and Inline BPEL Subprocess Invocations and Differences Between Oracle SOA Suite Templates and Reusable Subprocesses.

Note:

- There is no restriction on one BPEL subprocess calling itself recursively. You must determine if you want to recursively call the same BPEL subprocess and the number of times the subprocess calls occur.
- You can create and successfully deploy a SOA composite application that contains only a standalone subprocess. For example, create a SOA composite application and add a standalone subprocess in which you define two parameters for the subprocess and define an assign activity in the subprocess to swap the values of both parameters. However, while a SOA composite application that contains only a standalone subprocess and no other components can be deployed, it has no practical purpose.
- A standalone subprocess cannot be shared in the MDS Repository. However, a BPEL process with call activities for calling the subprocess can be shared in the MDS Repository

---

How to Create a Standalone BPEL Subprocess

This section provides an example of how to create a simple application that uses a standalone subprocess.

Note:

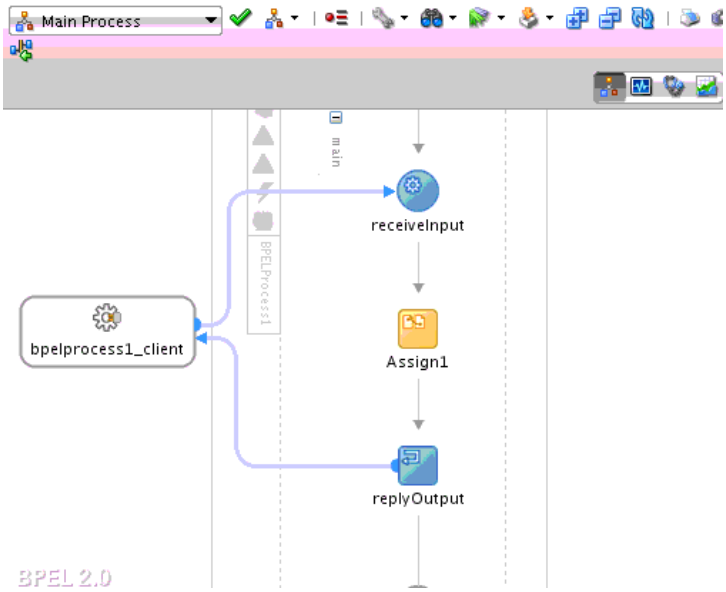
A standalone subprocess can include an inline subprocess.

To create a standalone BPEL subprocess:

1. Create a SOA composite application that includes a BPEL 2.0 process. For this example, a synchronous BPEL 2.0 process is created.
2. Design a BPEL 2.0 process. For this example, the following process is designed:
   - A variable of type string is created (for this example, named `variable1`) to pass in as a parameter.
   - An assign activity is created in which the client input string is mapped to `variable1`.

   Figure 39-21 shows the BPEL process design.

Figure 39-21 BPEL 2.0 Process Design



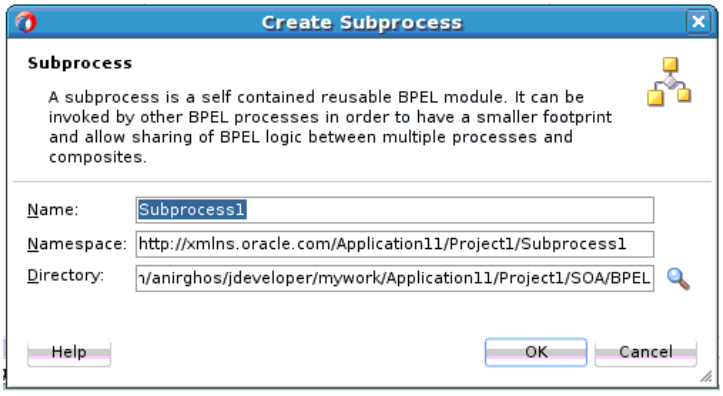
3. Click the *composite_name* link above Oracle BPEL Designer to access the SOA Composite Editor.

4. Right-click inside the SOA Composite Editor, and select **Insert** > **Subprocess** or drag a **Subprocess** icon from Components window into the composite.

   The Create Subprocess dialog is displayed.

5. Enter appropriate values or accept the default values, and click **OK** to create the standalone subprocess. Figure 39-22 provides details. For this example, the subprocess name provided is `Subprocess1`.

Figure 39-22 Create Subprocess Dialog



6. Right-click the subprocess in the SOA Composite Editor, and select **Edit**.

7. Create a variable of type string in the subprocess (for this example, the variable is named **p1**), and click **OK**,

   You now design simple process logic in the standalone subprocess.

8. From the Components window, drag an **Assign** activity into the process.

9. In the **Target** section of the **Copy Rules** tab of the assign activity, drag the **Expression Builder** icon onto the **p1** variable.

10. Create a concat expression to read the value out of the parameter in the subprocess and update variable **p1** with that value.

    ```
    concat($p1,",from subprocess")
    ```

11. Save the composite or select **Save All**, and exit the BPEL 2.0 process.

12. In the SOA Composite Editor, right-click the BPEL process and select **Edit**.

13. From the **Oracle Extensions** subsection, drag a call activity below the assign activity in Oracle BPEL Designer.

14. Right-click the call activity and select **Edit**.

    This invokes the Edit Call dialog. Note that variable **p1** is displayed in the **Name** column after the selected **Subprocess1**.

15. Click inside the **Value** column to invoke the Variable Chooser dialog.

16. Select **variable1**, and click **OK**. This maps variable **p1** from the standalone subprocess to variable **variable1** of the initial BPEL 2.0 process that you created.

17. Leave the **Copy By Value** check box deselected.

    Leaving this check box deselected copies the variable by reference. Only variables or partner links are accepted for variables, not XPath function queries. Copy by reference supports both input and output variables. Copy by value supports only input values.

18. From the Components window, drag a second **Assign** activity below the call activity.

19. In the **Copy Rules** tab of the assign activity, update the output message with **variable1**, and click **OK**. Figure 39-23 provides details.

Figure 39-23 Edit Assign Dialog

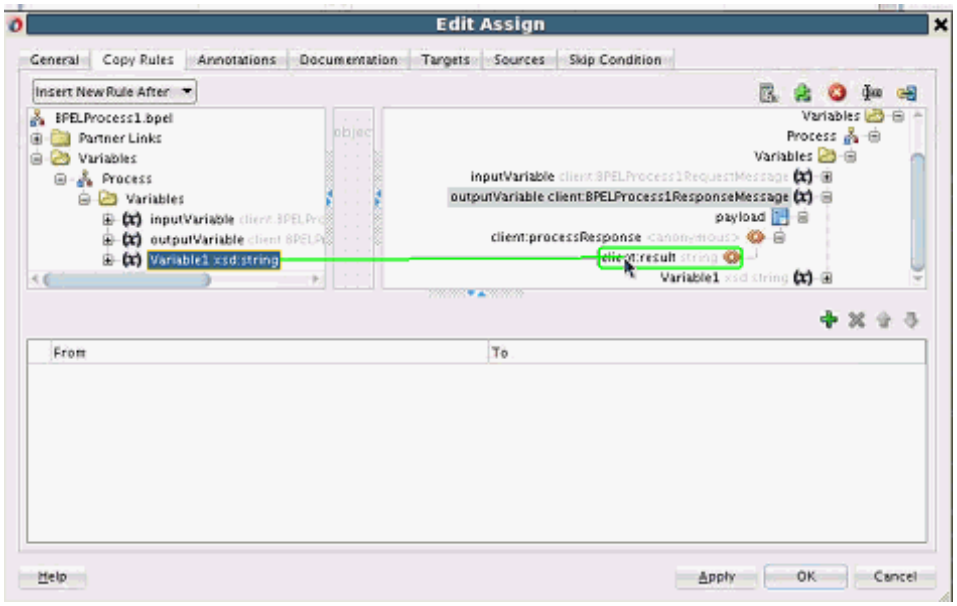
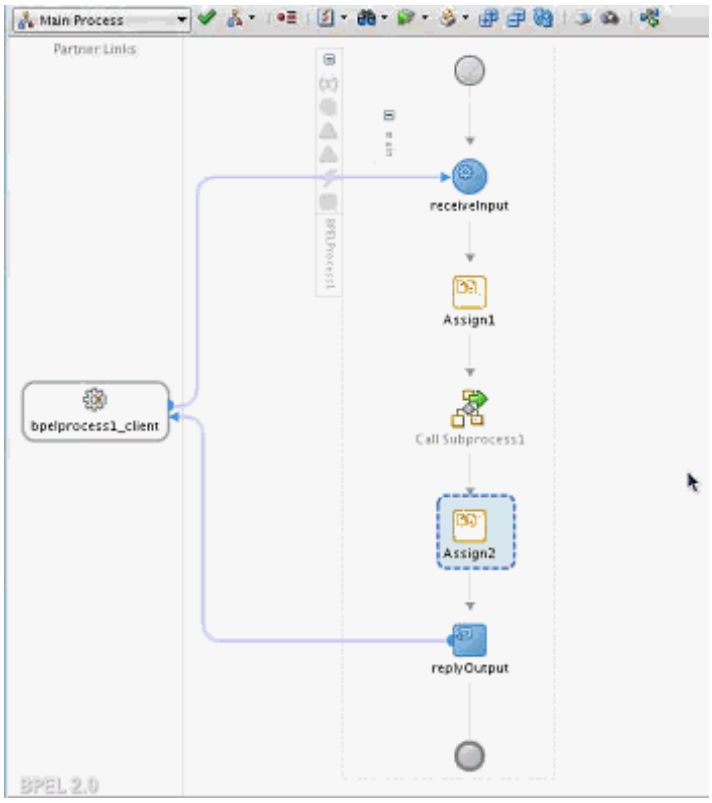


Figure 39-24 shows the BPEL 2.0 process with the subprocess. In this BPEL 2.0 process, the following logic is designed:

- The string value in the input message in **Assign1** is taken and assigned to **variable1** in the **call** activity, to be passed by reference.
- **assign2** takes **variable1** and creates the response. The **variable1** value is updated by the subprocess.

Figure 39-24 BPEL 2.0 Process



20. Go to the SOA Composite Editor and note that the BPEL subprocess is now connected to the BPEL 2.0 process because of the call activity.

    You are now ready to deploy the SOA composite application and create a business flow instance in Oracle Enterprise Manager Fusion Middleware Control.

    When you access the audit trail for the created business flow instance in Oracle Enterprise Manager Fusion Middleware Control, note that the call activity and its contents are displayed.

    For more information about standalone BPEL subprocesses, see Section "Using Templates and Standalone Subprocesses to Update the Order Status in the Database" of *Understanding Oracle SOA Suite*.

How to Create an Inline Subprocess

An inline subprocess is similar to a standalone subprocess, except that the inline subprocess is embedded in the parent process. For example, you may have a BPEL 2.0 process that includes assign and invoke activities within a scope activity that update the status of a customer order. You may have a business need for repeating these same activities later in the same process. One method is to physically repeat the same assign

and invoke activities of the scope activity later in the process, but this can be error prone. In addition, every time a change is necessary, it must be implemented in both scopes. As an alternative to repeating the activities, you can use an inline subprocess.

Note:

Creating an inline subprocess within an existing inline subprocess is not supported.

To create an inline subprocess:

1. Go to the scope activity in the BPEL 2.0 process that includes the assign and invoke activities that update the status of a customer order.

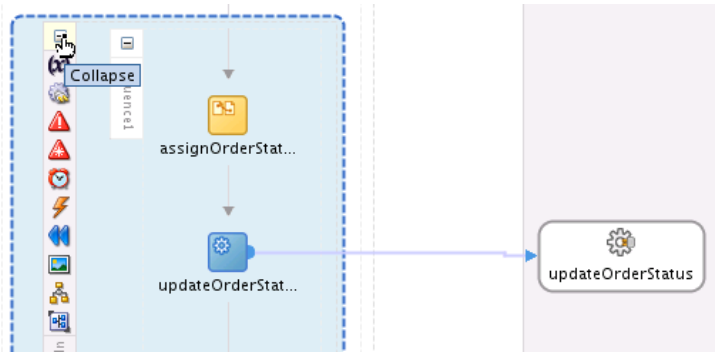
Note:

Inline subprocesses can also be created in a BPEL process by selecting **Inline Subprocesses** from the **Property Structure** menu above Oracle BPEL Designer, selecting the **Inline Subprocesses** folder, and clicking **Add**.

---

2. Collapse the scope activity. Figure 39-25 provides details.

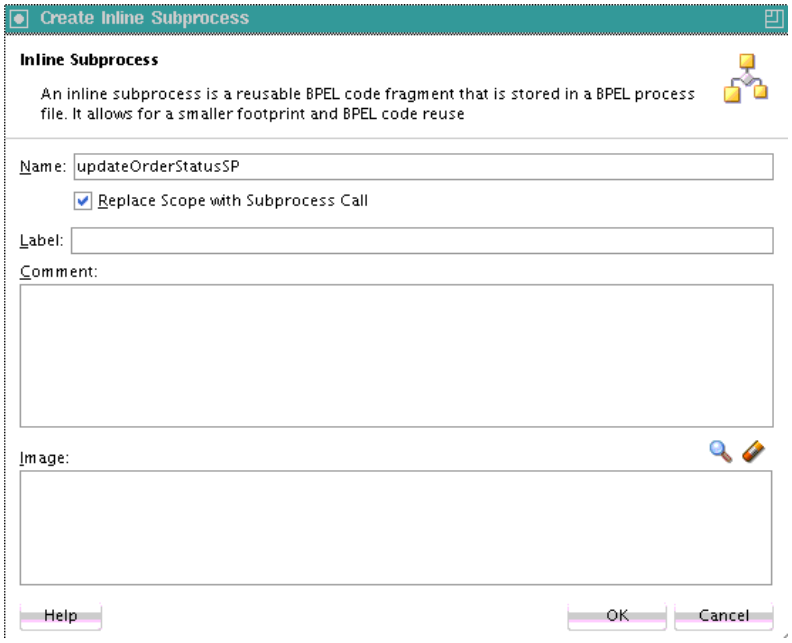
Figure 39-25 Scope Activity



3. Right-click the scope activity, and select **Convert to a Subprocess**.

   The Create Inline Subprocess dialog is displayed, as shown in Figure 39-26.

Figure 39-26 Create Inline Subprocess Dialog



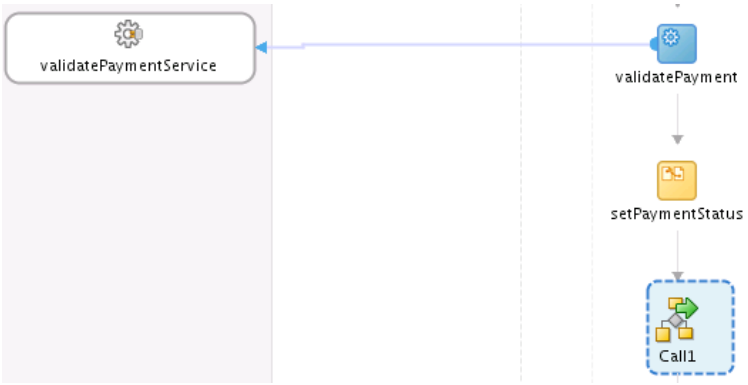
4. Enter values appropriate to your environment, then click **OK**.

Table 39-4 Create Inline Subprocess Dialog

Element	Description
Name	Enter a name or accept the default value, which defaults to the scope name.
Replace scope with subprocess call	Select to automatically replace the scope with a BPEL call activity (the default selection). If you want to create an inline subprocess and keep the selected scope in the process, you can deselect this check box.
Label	Optionally enter a description.
Comment	Optimally enter a comment.
Image	Select to replace the standard call activity icon with a unique image.

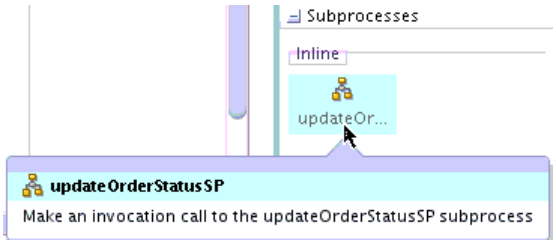
The scope activity is converted to a call activity in the BPEL 2.0 process, as shown in Figure 39-27.

Figure 39-27 Call Activity



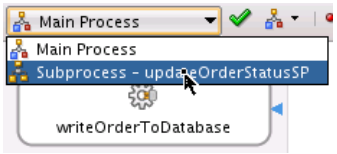
The new inline subprocess is also displayed in the **Subprocess** section of the Components window. Figure 39-28 provides details.

Figure 39-28 Inline Subprocess in Components Window



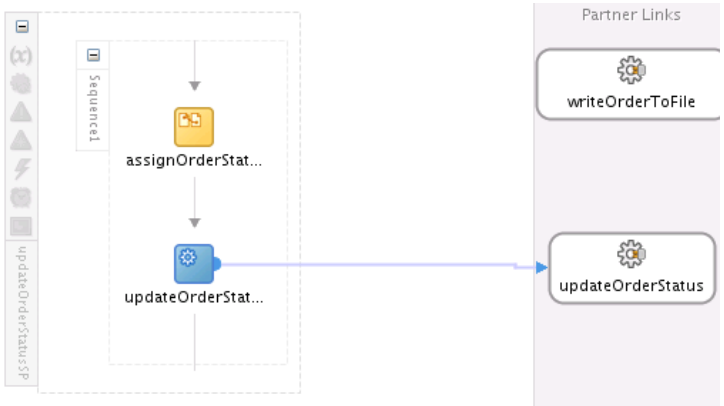
5. Above Oracle BPEL Designer, select **Subprocess - updateOrderStatusSP** to display the contents of the subprocess (the same contents as the initial scope activity). Figure 39-29 provides details.

Figure 39-29 Subprocess Selection Above Oracle BPEL Designer



The contents of the inline subprocess are displayed. Figure 39-30 provides details.

Figure 39-30 Inline Subprocess Contents

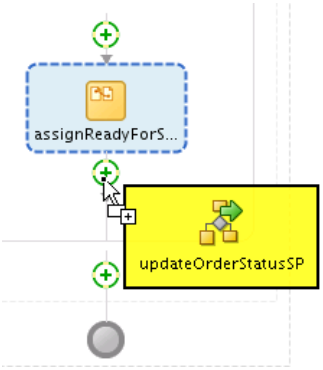


6. Make changes to the subprocess, if required, such as adding additional invoke activities.

   You can add the subprocess to the same BPEL 2.0 process, as necessary.

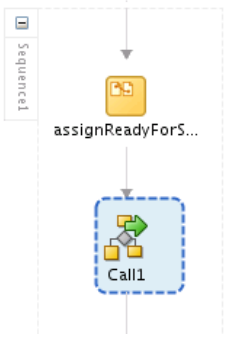
7. From the **Subprocess** section of the Components window, drag the inline subprocess into an appropriate section of the BPEL 2.0 process. Figure 39-31 provides details.

Figure 39-31 Subprocess Added to Same BPEL 2.0 Process



The subprocess name is automatically changed to **Call***number* as shown in Figure 39-32.

Figure 39-32 Subprocess Name Changed



For more information about using inline BPEL subprocesses, see Section "Updating Order Status with an Inline BPEL Subprocess" of *Understanding Oracle SOA Suite*.

How to Create a Standalone Subprocess that Takes a Partner Link as a Parameter

This section describes how a subprocess takes a partner link as a parameter and uses it to call the partner and return the result. You are essentially using a partner link from subprocess to subprocess.

To create a standalone subprocess that takes a partner link as a parameter:

1. Create a SOA composite application that includes a BPEL 2.0 process. For this example, a synchronous BPEL 2.0 process is created.
2. Go to the SOA composite application in the SOA Composite Editor.
3. Right-click and select **Insert** > **Subprocess**.

The Create Subprocess dialog is displayed.

4. Accept the default values (for this example, the default name is `Subprocess1`), and click **OK**.

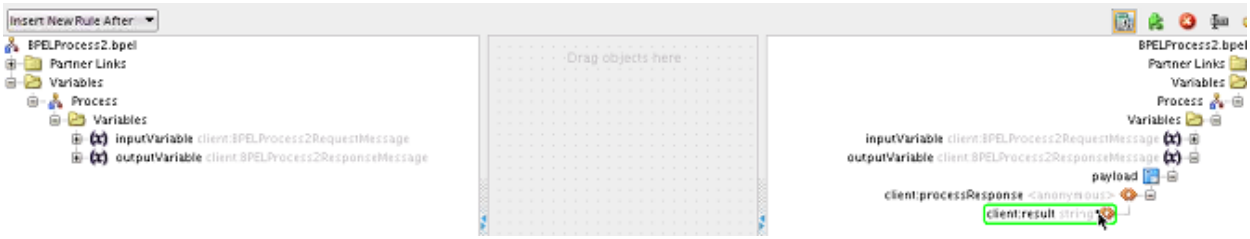
   You now create a second process to use as the partner link.

5. Create a second synchronous BPEL 2.0 process in the SOA composite application for this example, named `BPELProcess2`). This is the process to call.

6. From the Components window, drag an **Assign** activity into the second BPEL 2.0 process.

7. In the **Target** section of the **Copy Rules** tab, drag the **Expression Builder** icon onto the **result** variable. Figure 39-33 provides details.

Figure 39-33 Edit Assign Dialog



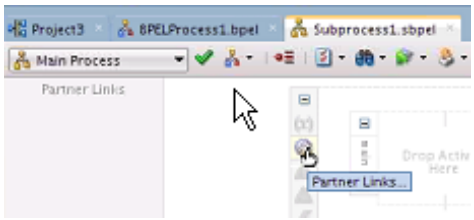
8. Build an XPath expression, and click **OK**.

   ```
   string("hello from process2")
   ```

9. Save the second BPEL 2.0 process, and return to the subprocess.

10. Click the **Partner Links** icon, as shown in Figure 39-34.

Figure 39-34 Partner Link Creation



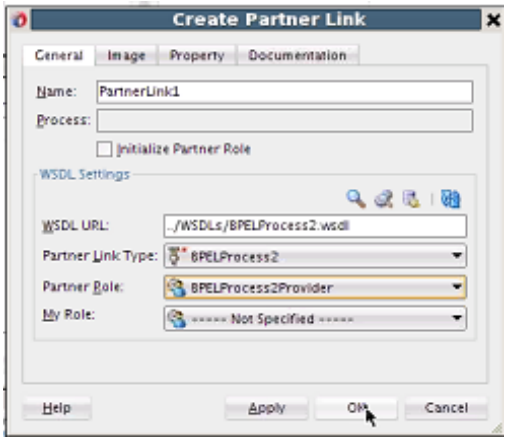
The Partner Links dialog is displayed.

11. Click the **Add** icon.

    The Create Partner Link dialog is displayed. You now define this partner link as a parameter.

12. Design the partner link (for this example, named `PartnerLink1`), and click **OK**. Figure 39-35 provides details. The role of the partner link is as the provider.

Figure 39-35 Partner Link Creation

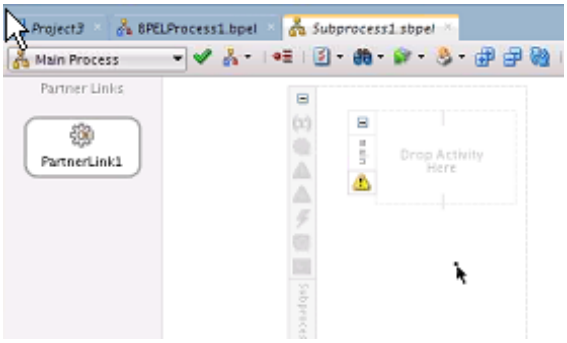


The Partner Links dialog looks as shown in Figure 39-36.

Figure 39-36 Partner Links Dialog

Figure 39-37 shows the contents of the subprocess.

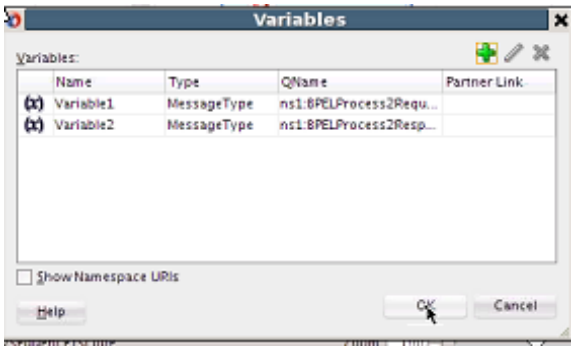
Figure 39-37 Subprocess Contents



13. Drag a **Scope** activity into the subprocess.

14. Click the **Variables** icon in the scope activity, and create request and response message type variables. Figure 39-38 provides details.

Figure 39-38 Request and Response Message Type Variable Creation



15. Drag a **Sequence** activity into the subprocess.
16. Drag an **Invoke** activity into the subprocess for invoking the partner link.
17. Design the invoke activity to invoke the partner link in the subprocess, as shown in Figure 39-39. The design includes the output variable (**Variable2**).

Figure 39-39 Edit Invoke Dialog

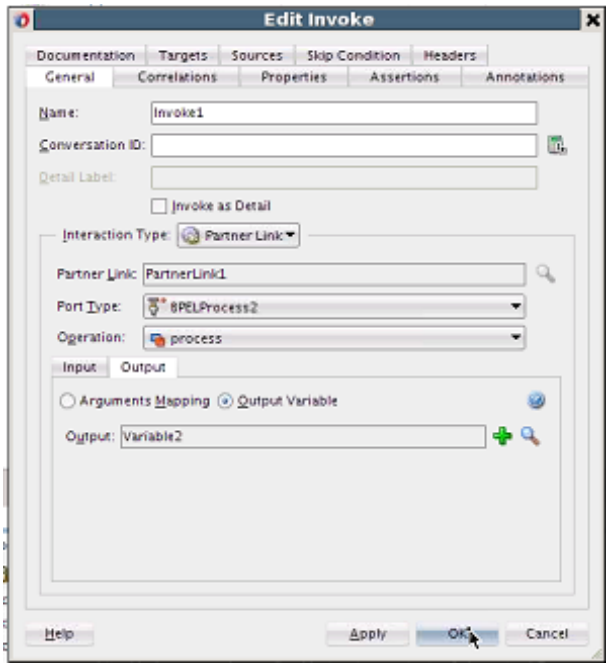
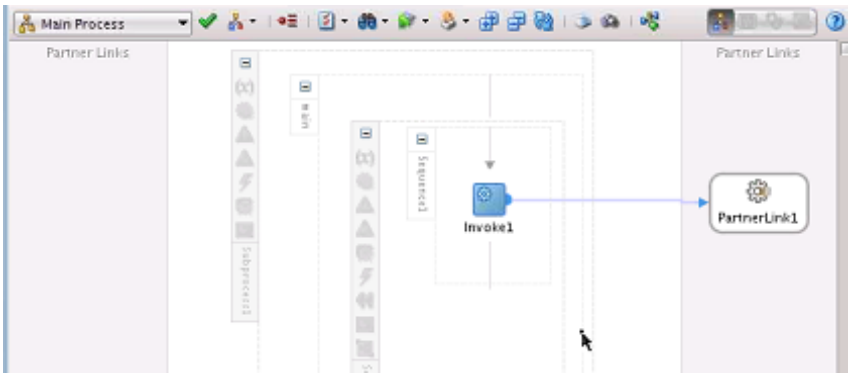


Figure 39-40 shows the subprocess.

Figure 39-40 BPEL Subprocess



18. Click the **Variables** icon in the subprocess to create a string variable to return the result.

    The Variables dialog is displayed.

19. Click the **Add** icon to invoke the Create Variable dialog.

20. Create a string variable (for this example, named `result`).

21. Drag an **assign** activity into the subprocess.

22. Map the result of the partner link invocation to the **result** variable, and click **OK**, as shown in Figure 39-41.

Figure 39-41 Edit Assign Activity

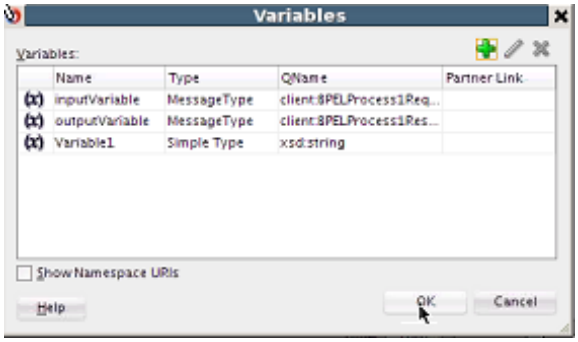
Subprocess design is now complete.

23. Return to the main BPEL 2.0 process in Oracle BPEL Designer (**BPELProcess1**).

24. Click the **Variables** icon in the process.

25. Click the **Add** icon to create a string variable to contain the result configured in Step 22 and passed back (for this example, named **Variable1**). Figure 39-42 provides details.

Figure 39-42 Variables Dialog

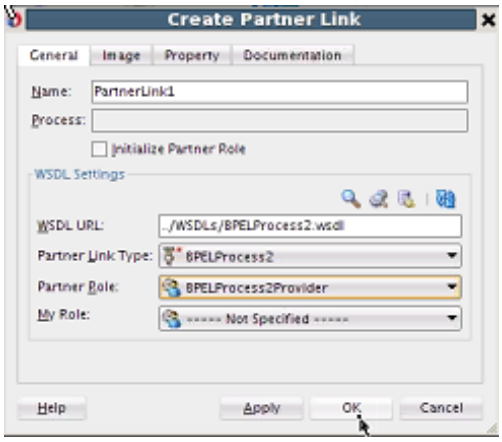


26. Add an assign activity to assign the string value to **Variable1**.
27. Drag a **call** activity below the assign activity in Oracle BPEL Designer. Figure 39-43 provides details.

Figure 39-43 Subprocess Added to Main BPEL 2.0 Process

28. Right-click the **Partner Links** swimlane, and select **Create Partner Link**.
29. Design a partner link to invoke **BPELProcess2**, as shown in Figure 39-44.

Figure 39-44 Create Partner Link Dialog



30. Right-click the **Call** activity, and click **Edit**.

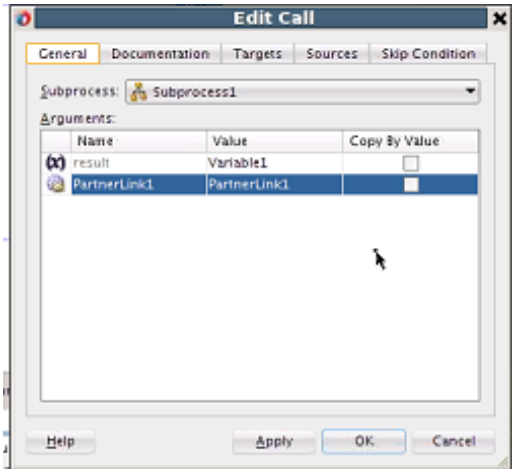
    The Edit Call dialog shows the partner link created earlier in the subprocess.

31. In the **result** row, click the **Value** column to invoke the Variable Chooser dialog.
32. Select **Variable1**, and click **OK**.

33. In the **PartnerLink1** row, click the **Value** column to invoke the Partner Link Chooser dialog.

34. Select **PartnerLink1**, and click **OK**. Figure 39-45 shows the Edit Call dialog with design complete. Since the variables are sent by reference, if the subprocess does something to change the partner link (such as copying in another partner link), that impacts the calling process's partner link. This is the same process as with variables.

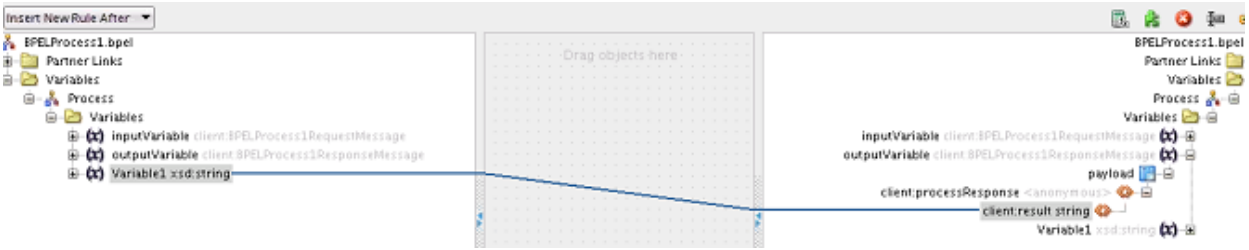
Figure 39-45 Edit Call Dialog



35. Drag an **Assign** activity below the **Call** activity to return the result.

36. In the **Copy Rules** tab, map **Variable1** to **result** to return the result to the caller. Figure 39-46 provides details.

Figure 39-46 Edit Assign Dialog



37. Deploy the SOA composite application.

What You May Need to Know About Renaming a Subprocess

When you rename a subprocess, it is not updated in the invoking call activity. You must manually update the subprocess name in the call activity.

Assume you perform the following steps:

1. Create an asynchronous BPEL 2.0 process.
2. Right-click the SOA Composite Editor, and select **Insert** > **Subprocess**.
3. Create a subprocess named **SubProcessNew**.
4. Right-click **SubProcessNew**, and click **Edit**.

5. From the Components window, drag an **Empty** activity into the subprocess.
6. Open the asynchronous BPEL 2.0 process.
7. From the Components window, drag a **Call** activity into the process.
8. Invoke the **SubProcessNew** subprocess from the call activity.
9. Return to the SOA Composite Editor, and rename the **SubProcessNew** subprocess to **SubProcessRenamed**.
10. Open the call activity in the asynchronous BPEL 2.0 process, and note that the **Subprocess** field is now empty.
11. In the **Subprocess** field, manually enter the updated name of **SubProcessRenamed**.

40

Creating Transformations with the XSLT Map Editor

This chapter describes how to use the XSLT Map Editor to create, design, and test data transformations between source schema elements and target schema elements.

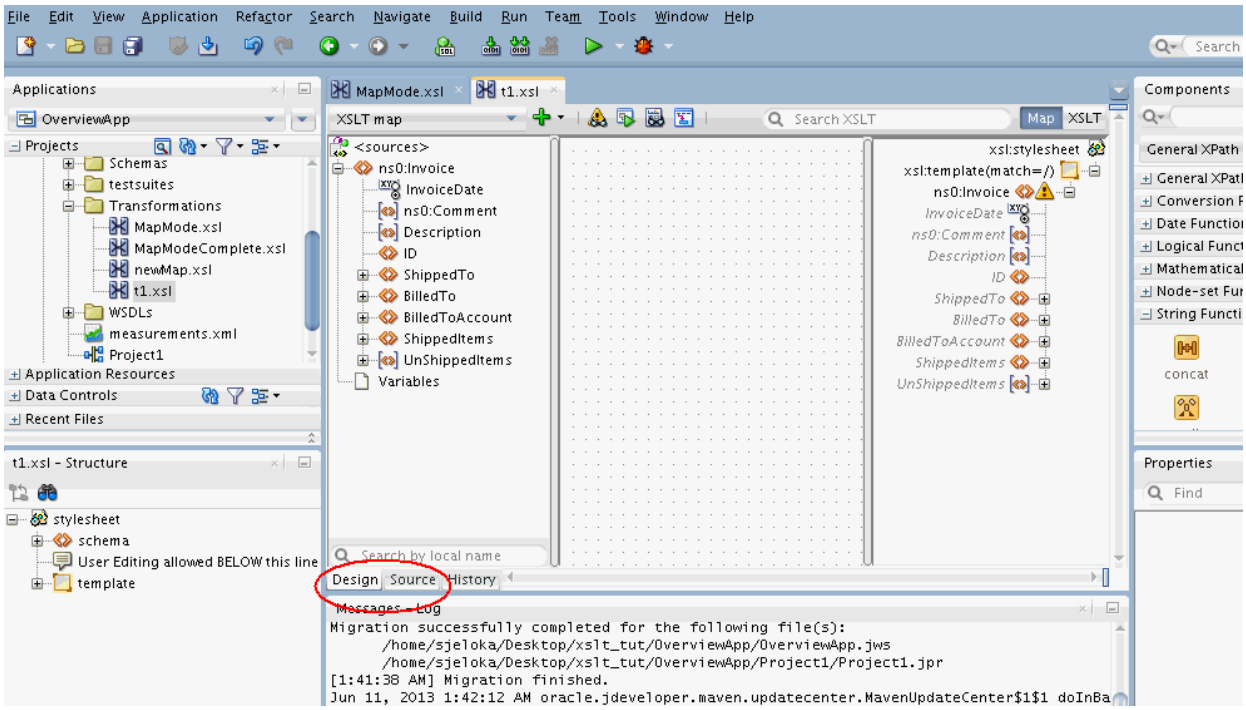
This chapter includes the following sections:

Introduction to the XSLT Map Editor

The XSLT Map Editor enables you to edit XSLT stylesheets using a graphical editor. It also provides the feature to directly edit the XSLT source.

Figure 40-1 shows the XSLT Map Editor. You can switch between the graphical editor and the source view using the tabs at the bottom of the editor. Click **Design** to edit using the graphical editor. Click **Source** to edit using the source editor.

Figure 40-1 XSLT Map Editor



You can move back and forth between the **Source** and **Design** tabs. Any change made under one tab is reflected in the other tab. A **History** tab is also available to enable you to view the revision history, and revert to any point in the edit history.

The XSLT Map Editor fully supports XSLT 1.0 and XPath 1.0.

If you want to use XSLT 2.0, then you can change the XSLT version in the source view.

All XSLT 2.0-specific constructs must be added in the source view. You can then choose to switch to the design view, and continue to edit the map. XSLT 2.0-specific constructs are shown in design view, but can be modified only in the source view. XPath 2.0 constructs can also be added in the design view. However, XPath 2.0 constructs are not parsed into separate graphical elements in the design view. You must edit the full XPath statement in text form.

The XSLT Map Editor provides the following edit views under the design view:

- Map View
- XSLT View

You can switch between the two views using the buttons at the top right hand corner of the XSLT editor. Click **Map** to use the traditional Map View of the XSLT editor. Click **XSLT** to use the XSLT View for more complex XSLT maps.

Using the Map View

Figure 40-1 shows the Map View of the XSLT Map Editor. The left pane contains the source tree representing the incoming source XML document. The source tree can be created from an XSD schema file or a sample XML file.

The center pane, or the canvas, is the place where you drop XPath expressions and functions that can be mapped to XSLT elements.

The right pane is the target pane representing a merged view of the XSLT being created, and the target tree that represents the target schema. The target tree can be created from an XSD schema file or a sample XML file.

The grayed nodes, in italics, in the target tree represent nodes that haven't been mapped yet. These nodes are not part of the XSLT, and are displayed for convenience. Once a grayed node is mapped, it appears in regular font, and gets represented in the XSLT map.

Map View supports drag-and-drop mappings from source tree to target tree. Map View also supports XPath function calls and XSLT statements such as `xsl:if` and `xsl:for-each.`

As Map View does not separate the XSLT statements from the target tree, it is limited to the following:

- Only one XSLT template rule with the `match='/'` attribute is supported.
- The following XSLT statements are supported: `xsl:for-each, xsl:value-of, xsl:text, xsl:if, xsl:choose/when, xsl:variable and xsl:param.`

Use the XSLT View for complex XSLT statements that require separating the XSLT statements from the target tree.

Using the XSLT View

The XSLT View is a more advanced mode that enables you to separate the XSLT statements from the target tree document. This enables you to create complex XSLT statements without leaving the design view. Source and target schemas are optional in the XSLT View.

The XSLT View includes the same panes as the Map View, except that the right target pane is divided into two panes. The top pane is called the XSLT pane and the lower pane is called the target pane. If no target schema is defined, then the lower pane is not shown. If no source schema is defined, the source pane is still displayed to enable you to add parameters and variables, whose values can be referenced by the XSLT.

In XSLT View, you can create any series of XSLT statements without having to intersperse these statements around target tree nodes. For instance, in the 11*g* mapper, all `xsl:if` statements had to contain a single target output node. In XSLT View, the `xsl:if` statement can be used anywhere, and can contain any other XSLT statement.

The XSLT View supports all XSLT 1.0 statements. The XSLT View also supports multiple template rules with or without source and target schemas. The XSLT View enables you to graphically display and edit any XSLT stylesheet, irrespective of the complexity involved.

Using the Components Window

The Components window contains all the XPath functions and XSLT elements and templates that you can use in your XSLT map.

The Components window is located at the upper right-hand corner of Oracle JDeveloper, by default. If the Components window does not appear, click **Components** under the **Window** menu to display the Components window. You can optionally choose to drag the Components window to any convenient location in the JDeveloper window. You can also resize the Components window, as desired.

The Components window organizes these functions, elements, and templates under the following categories:

- Advanced XPath:
- General XPath:
- XML:
- XSLT Elements:
- XSLT Templates:
- All Pages:
- User Defined:
- My Components:

Using the Properties Window

The Properties window shows the content and properties of the item selected in the XSLT Map Editor. Some of these properties can also be edited.

The Properties window is located below the XSLT Map Editor, by default. If the Properties window does not appear, click **Properties** under the **Window** menu to display the Properties window. You can optionally choose to drag the Properties window to any convenient location within the JDeveloper window. You can also resize the Properties window, as desired.

The Properties window, in general, can be used to display and edit the properties of the following items:

Selected Element in Editor	What is Shown in Properties Window	Whether Editable (Yes/No)
Source tree node	Schema Information for the selected element or attribute.	No
Target tree node	Schema information for the selected element or attribute.	No
XSLT tree node: XSLT element	XSLT element attributes and their values	Yes
XSLT tree node: literal element or attribute	Literal element or attribute name and namespace	Yes
XPath expression folder in Canvas pane	Full text XPath expression	Yes
Function icon within expression folder in Canvas pane	XPath field for each parameter of the function	Yes
Line connecting source and target node	Full text XPath expression	Yes

Creating an XSLT Map

XSLT maps can be created from scratch, or from other editors such as BPEL, BPM, and Mediator.

How to Create an XSLT Map

To create an XSLT Map:

1. From the File main menu, select **New** > **XSL Map.** Alternatively, right-click the project folder and select **New** > **XSL Map.**

   The Create XSL Map File dialog appears.

2. Under **File Name,** specify a name for your .xsl map file.
3. Under **Directory name,** select the destination directory for the .xsl file.
4. Under Sources, select **Use Source Schema(s)** to specify a source schema for the map.
5. Under Primary Source, click **Browse** to select the source schema. The Select Schema dialog box appears.
6. Choose **Select Schema** if you want to use an XSD schema file or WSDL file for the source schema.

   Note:

   You can alternatively use a sample XML file as the schema source.

   Select **Generate from XML** to generate the schema from an XML file. Select the sample file and click **Open.** Go to Step 9.

   ---

7. Click **Browse** to select a schema file and element for the source schema. The Type Chooser dialog appears.
8. Select the schema file and the corresponding element from the project schema files or project WSDL files tree. Click **OK.**

   If the schema or wsdl file that you need is not available in the tree, you may import a schema or wsdl file by clicking the **Import Schema File** or **Import WSDL File** button at the top right corner of the dialog.

9. Click **OK** in the Select Schema dialog.

   Note:

   Under Additional Sources, you can click the Add Schema button identified by the **green plus icon** (+) to add any additional sources in the form of parameters.

   ---

10. Select **Use target schema** to specify a target schema for your XSL map.

11. Click **Browse** to select the target schema. The Select Schema dialog appears.

12. After selecting the target schema, click **OK** in the Select Schema dialog.

Note:

When a Target Schema is used, initial element and attribute nodes may be generated in the XSLT pane depending upon the current Preferences setting.

The default setting is to generate a root template with a `match=''/''` attribute followed by all required elements and attributes in the target schema.

---

13. Click **OK** to create the XSL map file.

Note:

- Once the XSLT map is created you may add or replace source and target schemas by selecting the appropriate option from the context menu in the canvas pane.

  For example, you may add additional sources as parameters by selecting **Add Parameter** from the context menu on the source pane.

- You may edit a source or target schema file that is being used by an XSLT Map, using JDeveloper. Upon saving the schema file, the source or target tree in the XSLT editor is automatically updated.

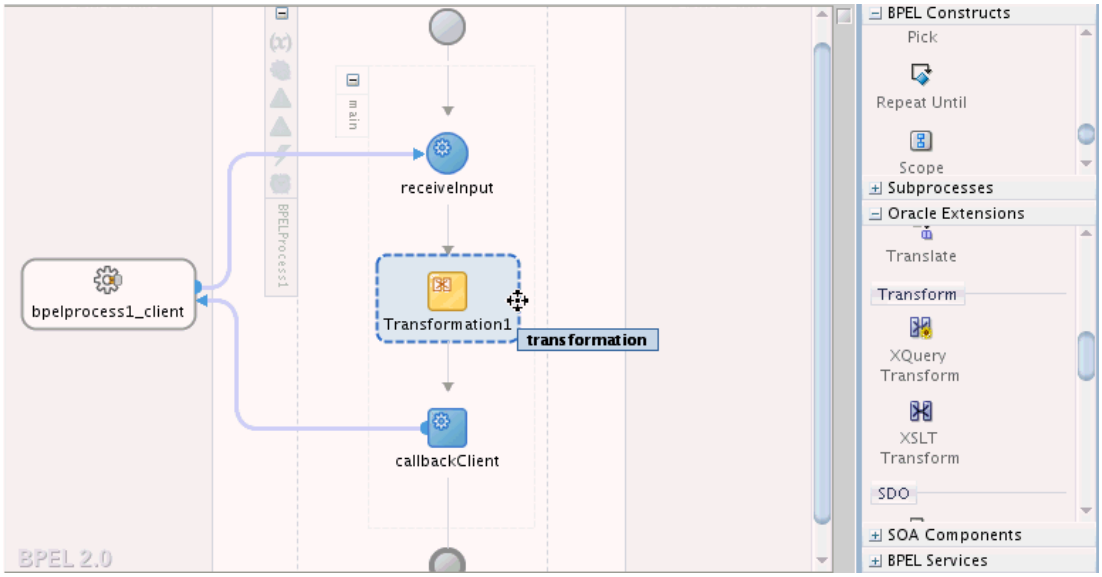
How to Create an XSL Map File in Oracle BPEL Process Manager

An XSLT Transform activity enables you to create a transformation using the XSLT Map Editor 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 Components window, drag an **XSLT Transform** activity into your BPEL process diagram. Figure 40-2 provides an example.

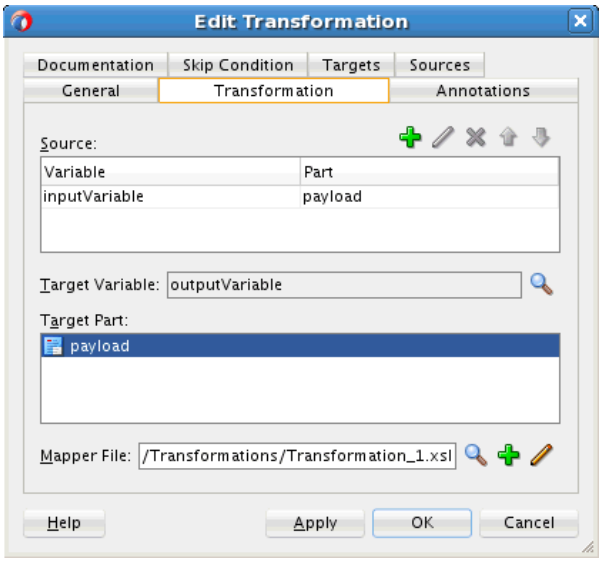
Figure 40-2 Transform Activity



2. Double-click the **XSLT Transform** activity.

   The Transform dialog shown in Figure 40-3 appears.

Figure 40-3 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.

Note:

Figure 40-3 shows the Edit Transformation dialog for BPEL 2.0. The Edit Transformation dialog for BPEL 1.1 is slightly different. In the Edit Transformation dialog for BPEL 1.1, you can select the **Target Variable** from the list of variables.

---

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. You create your mappings in the map file using the XSLT Map Editor.

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 Map Editor appears.

Note:

If you select a file with a `.xslt` extension such as `xform.xslt`, it opens the XSLT Map Editor to create an XSL file named `xform.xslt.xsl`, even though your intention was to use the existing `xform.xslt` file. A `.xsl` extension is appended to *any* file that does not have a `.xsl` extension, and you must create the mappings in the new file. As a work around, ensure that your files first have an extension of `.xsl`. If the XSL file has an extension of `.xslt`, then rename it to `.xsl`.

---

6. Go to Introduction to the XSLT Map Editor for an overview of using the XSLT Map Editor.

How to Create an XSL Map File from Imported Source and Target Schema Files in Oracle BPEL Process Manager

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:

   a. In the Structure window of Oracle JDeveloper, right-click **Schemas**.

   b. 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**.

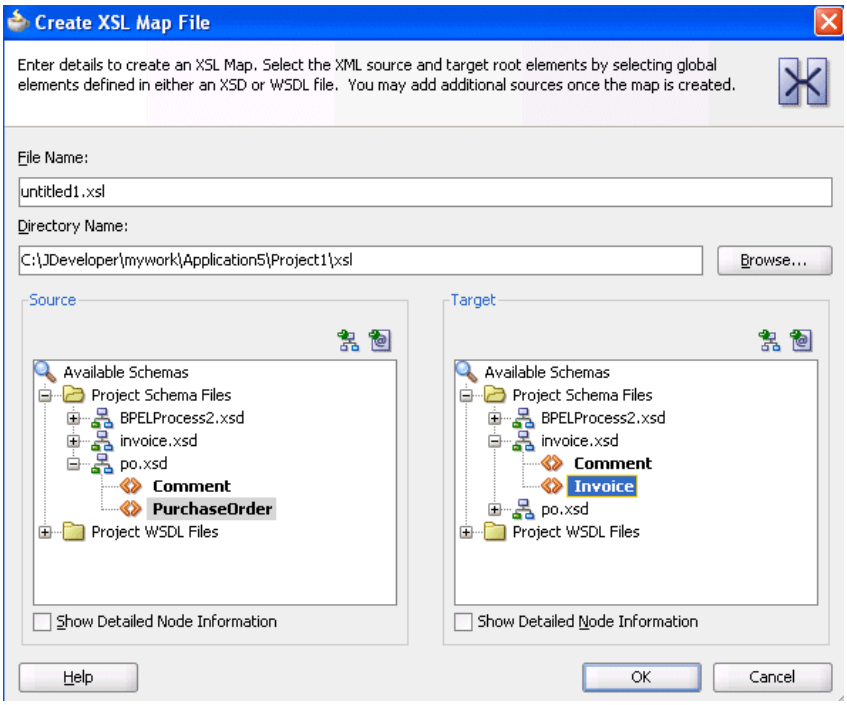
   The Create XSL Map File dialog appears. This dialog enables you to create an XSL map file that maps a root element of a source schema file or Web Services Description Language (WSDL) file to a root element of a target schema file or WSDL file. Note the following details:

   - – 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. In the **File Name** field, enter a name for the XSL map file.

7. Select the root element for the source and target trees. In the example in Figure 40-4, the **PurchaseOrder** element is selected for the source root element and the **Invoice** element is selected for the target root element.

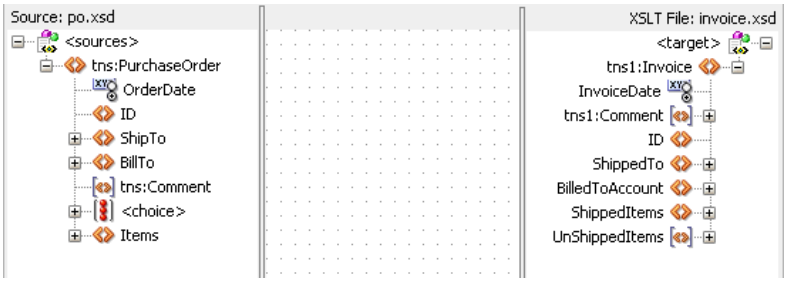
Figure 40-4 Expanded Target Section



8. Click **OK**.

   A new XSL map is created, as shown in Figure 40-5.

Figure 40-5 New XSL Map



9. Save and close the file now or begin to design your transformation. Information on using the XSLT Map Editor is provided in Introduction to the XSLT Map Editor.

10. From the Components window, 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 Map Editor displays your XSL map file.

16. Go to Introduction to the XSLT Map Editor for an overview of using the XSLT Map Editor.

How to Create an XSL Map File in Oracle Mediator

The XSLT Map Editor 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 Map Editor.

The XSLT Map Editor is available from the Applications window 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.

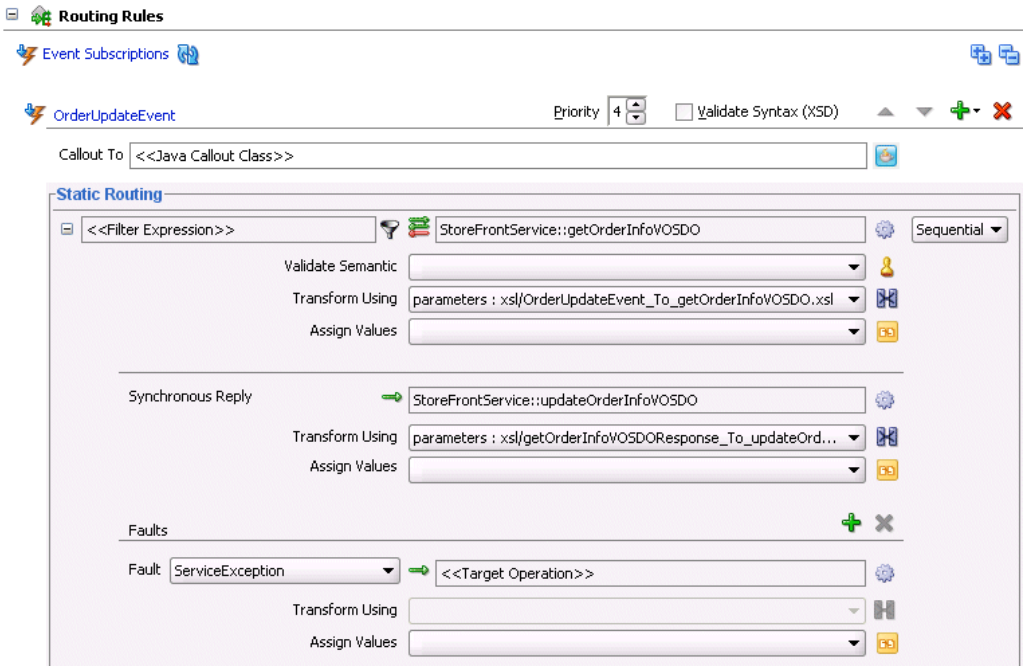
To launch the XSLT Map Editor from the Mediator Editor and create or update a data transformation XSL file, follow these steps.

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 40-6, click the appropriate **transformation map** icon to open the Transformation Map dialog.

Figure 40-6 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 40-7 appears.

Figure 40-7 Reply Transformation Map Dialog

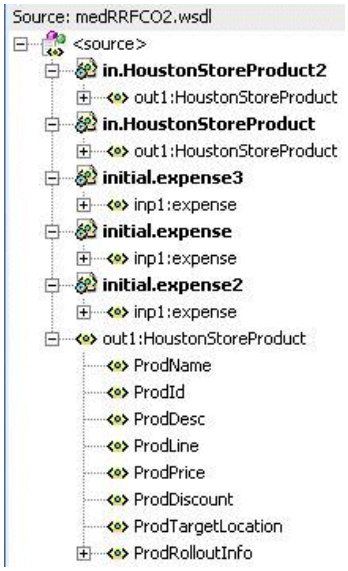
If the routing rule includes a synchronous reply or fault, the Reply Transformation Map dialog or Fault Transformation Map dialog contains the **Include Request in**

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 time-out 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 40-8 provides an example.

Figure 40-8 Reply Part

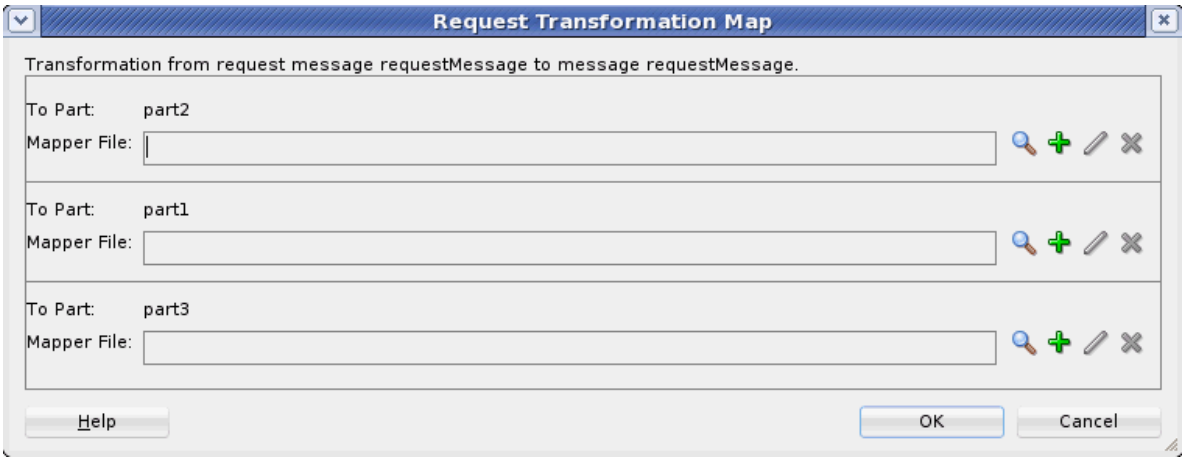


4. Choose one of the following options:

   - Click the **Search** icon to browse for an existing XSLT map file (or accept the default value).
   - Click the **Add** icon, to create a new XSLT map 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, Oracle Mediator's transformation dialog has a separate panel for each target part. For example, Figure 40-9 shows a request in which the target has three parts:

Figure 40-9 Request Transformation Map Dialog



5. Click **OK.**

   If you chose to create a new XSLT map, the XSLT Map Editor opens to enable you to correlate source schema elements to target schema elements.

6. Go to Introduction to the XSLT Map Editor for an overview of using the XSLT Map Editor.

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 Map Editor.

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, no errors are displayed during XSL transformation.

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 or assigning any values, Oracle Mediator passes the payload through.

However, for the payload to be passed through successfully, the source and target message part names must be the same, and of the same type. Otherwise, the SOA project fails to compile. For projects that have been upgraded from 11*g*, the project compiles, but the target reference may fail to execute with error messages such as `Input source like Null` or `Part not found`.

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 an Oracle 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.

Editing an XSLT Map in Map View

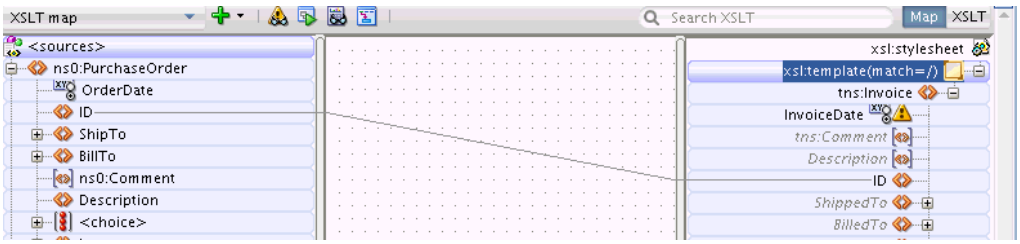
This section discusses basic functionality available in Map View. The remaining sections discuss editing in the XSLT View with notes on restrictions that might apply to Map View for the specific activity being discussed.

How to Perform a Value Copy by Linking Nodes

To copy the value of an attribute or leaf-element in the source to an attribute or leaf-element in the target, drag a line from the source node to the target node. A green highlighted line appears as you are dragging and dropping. When you complete the drop, a line is drawn connecting the source and target nodes.

Figure 40-10 shows the map view where the **PurchaseOrder/ID** source element is mapped to the **Invoice/ID** target element. Notice that a line connects the source and target nodes. Also, the ID element in the target tree is no longer grayed, and appears in normal font. This means that the ID element has been added to the XSLT map.

Figure 40-10 Copying a Leaf Node in Map View



How to Create an Empty Node in the Output Document

To create an empty node in the output document:

1. Select the grayed node in the target pane.
2. Right-click the node, and select **Create Node in XSLT** from the context menu.

How to Set a Literal Text Value for a Target Node

To set a literal text value on an output/target node:

1. Right-click the node in the target pane. Select **Edit Text Value** from the context menu that appears.

   The Set Text dialog appears.

2. Enter the text value to be assigned to the node. Do not enclose the text in quotation marks.
3. Click **OK.**

   A yellow T icon appears next to the node indicating that a text value has been set for the item. If you move the mouse cursor over the node, the corresponding text appears. If the node was grayed earlier, it no longer appears gray, as the node has been added to the XSLT map.

How to Add an XSLT Statement

You can add XSLT statements to handle constructs such as conditional statements (if-then-else) and iterations (for-each).

To Add an XSLT Statement:

1. Right-click the target node, and select **Add XSL Instruction** from the context menu that appears. A submenu appears with the various XSL statements that you can add.
2. Select the desired XSL statement, such as **if, choose,** or **for-each,** from the submenu.

The `xsl:text` and `xsl:variable` XSLT statements can only be added for existing nodes.

To Add an xsl:text or xsl:variable Statement:

1. Make sure that the target node exists in the XSLT.

   If the target node appears gray, right-click the target node and select **Create Node in XSLT** from the context menu that appears. The node no longer appears gray, and is added to the XSLT map.

2. Right-click the target node, and select **Add XSL Instruction** from the context menu that appears. A submenu appears with the various XSL statements that you can add.

3. Select **text** or **variable** from the submenu.

Note:

The `xsl:copy-of` statement is not supported in Map View. It is supported in XSLT View.

You can also choose to drag and drop XSLT statements from the Components window.

To Drag and Drop an XSLT statement to a Target Node:

1. Select the XSLT Elements page from the Components Window. A list of statement categories appear.

2. Locate a supported statement, for Map View, from a category. For example, the for-each statement appears under the Flow Control category.

   The Map View supports only a subset of XSLT statements. These statements are discussed individually in the sections that follow.

3. Drag the statement to the desired target node until green highlighting appears over the node, indicating that the statement can be dropped.

4. Drop the statement to insert it into the XSLT map.

The following sections enumerate the different XSLT statements that you can add using the map view:

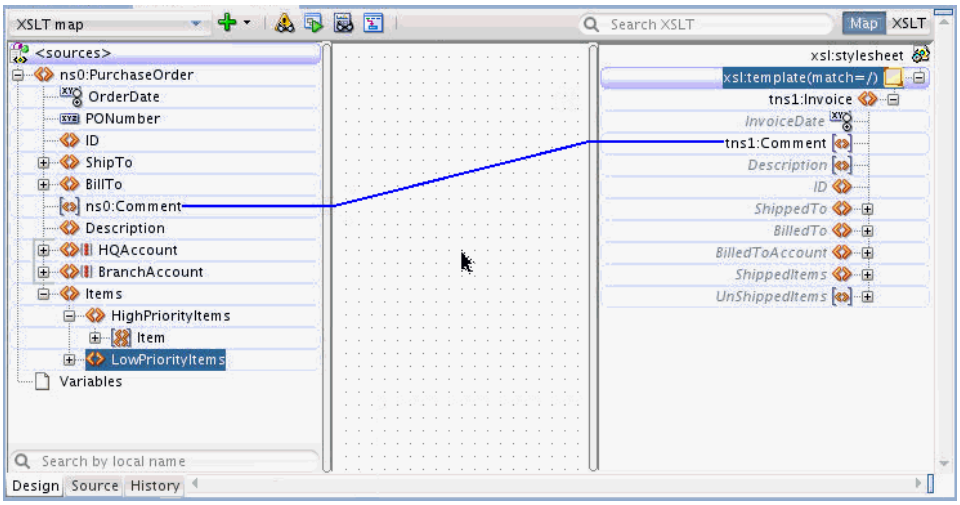
- How to Add Conditional Processing Using xsl:if
- How to Add Conditional Processing Using xsl:choose
- How to Add Loops Using xsl:for-each
- How to Add xsl:sort for an xsl:for-each Statement
- How to Duplicate XSLT Instructions

How to Add Conditional Processing Using xsl:if

If a source and target node are optional in their respective schemas, the `xsl:if` statement is often used to test for the existence of the source node before creating the corresponding target node.

In Figure 40-11, the Comment node is optional for both the source and the target. The square brackets around the Comment nodes indicate that they are optional nodes.

Figure 40-11 Optional Nodes in Source and Target Trees



If the source Comment node does not exist in the source document at runtime, its value is empty. This creates a Comment node in the target document and sets its value to empty.

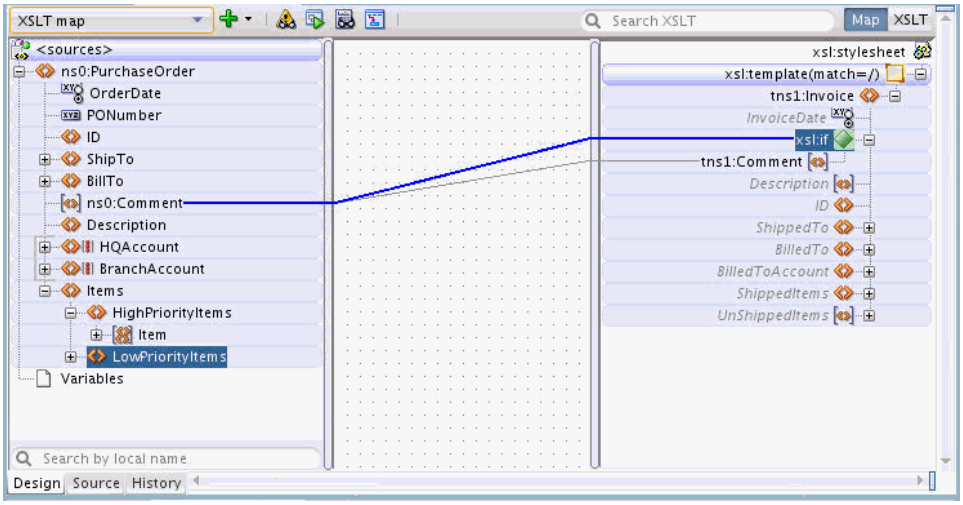
To prevent creating an empty node when the source node is not there, add an `xsl:if` statement above the target Comment node. The `xsl:if` statement tests for the existence of the source node before creating the target node.

To add an xsl:if statement using the context menu:

1. Right-click the target node and select **Add XSL Instruction -> if** from the context menu that appears. An `xsl:if` node is added as the parent node of the target node.

2. To set the condition for the `xsl:if` node, drag and drop the source node to the `xsl:if` node.

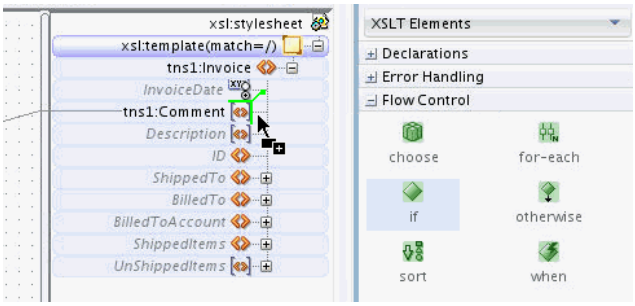
Figure 40-12 Dragging the Source Node to the xsl:if Node



To add an xsl:if statement using drag and drop:

1. In the Components window, select the XSLT Elements page.
2. Expand the Flow Control section. You can click the plus sign (+) next to Flow Control to expand the section.
3. Drag the **if** icon to the right side of the target node until you can see the green highlighting, as shown in Figure 40-13.

Figure 40-13 Adding an xsl:if Statement



4. Drop the **if** icon while the green highlighting is visible. An `xsl:if` node is added as the parent node of the target node.
5. To set the condition for the `xsl:if` node, drag and drop the source node to the `xsl:if` node.

When viewed in source view, the `xsl:if` statement looks similar to the following:

```
<xsl:if test="/ns0:PurchaseOrder/ns0:Comment">
    <tns1:Comment>
       <xsl:value-of select="/ns0:PurchaseOrder/ns0:Comment"/>
    </tns1:Comment>
</xsl:if>
```

The preceding `xsl:if` statement ensures that the target node is created only if the source node exists.

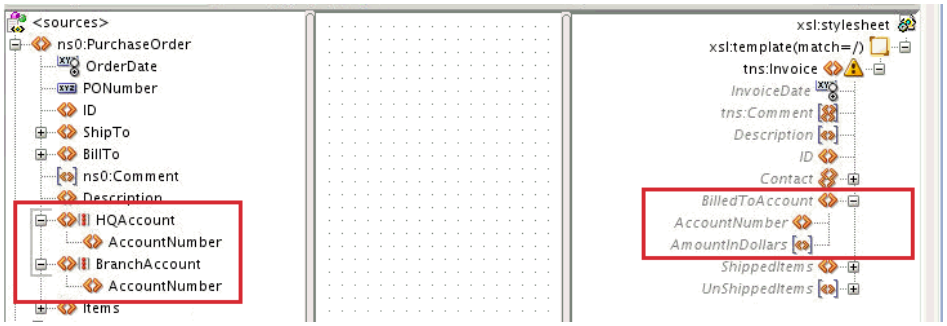
How to Add Conditional Processing Using xsl:choose

The `xsl:choose` statement is similar to the `xsl:if` construct. You can use the `xsl:choose` XSLT statement if there are multiple conditions to evaluate.

Figure 40-14 shows the XSLT Map Editor containing sample source and target schemas. The source schema has an xsd:choice construct defined. The source schema can contain either an HQAccount or a BranchAccount node, but not both. The target schema has a BilledToAccount/AccountNumber node that must be defined.

If the HQAccount node exists, you must copy its AccountNumber to BilledToAccount/AccountNumber in the target. If the HQAccount node does not exist, you must copy the AccountNumber from the BranchAccount node. You can use the `xsl:choose` statement to accomplish this task.

Figure 40-14 XSLT Map Editor Containing Sample Source and Target Schemas

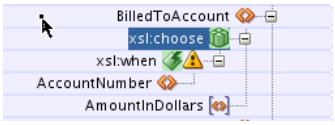


To add an xsl:choose statement using the context menu:

1. Right-click the target node and select **Add XSL Instruction -> choose** from the context menu that appears.

   A choose statement is added as the parent node of the target node along with an `xsl:when` statement. Figure 40-15 shows the result of adding the `xsl:choose` statement to the AccountNumber node.

   Figure 40-15 Adding an xsl:choose Statement

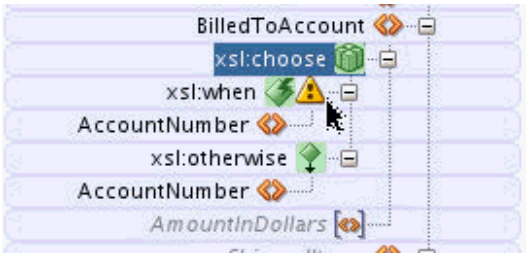
   

   An `xsl:choose` statement can contain multiple `xsl:when` statements followed by an optional `xsl:otherwise` statement.

2. To add an `xsl:otherwise` node to the xsl:choose node, right-click **xsl:choose** in the target tree and select **Add XSL Instruction -> otherwise** from the context menu that appears.

   Figure 40-16 shows the result of adding the `xsl:otherwise` statement to the `xsl:choose` statement. Note that the AccountNumber node is copied to each section of the `xsl:choose` statement.

   Figure 40-16 Adding an xsl:otherwise Statement to an xsl:choose Statement

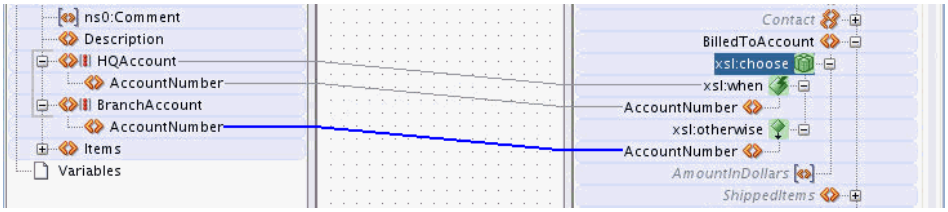
   

3. Map the `xsl:when` node to the source node whose existence is to be tested. In our current example, you drag a line from the **HQAccount** node in the source to the **xsl:when** node in the target.

4. Map the xsl:when and xsl:otherwise cases. In the current example, you drag a line from the **HQAccount/AccountNumber** node to the **xsl:choose/xsl:when/AccountNumber** node. Similarly, you drag a line from the **BranchAccount/AccountNumber** node to the **xsl:choose/xsl:otherwise/AccountNumber** node.

Figure 40-17 shows the completed `xsl:choose` construct.

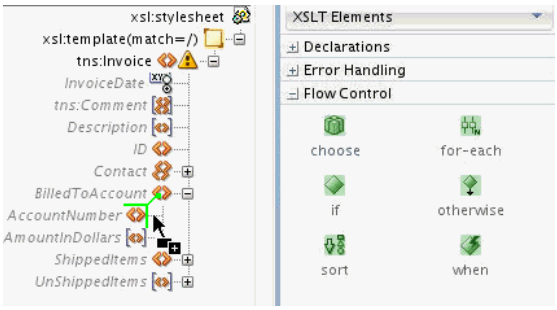
Figure 40-17 Sample xsl:choose Construct



To add an xsl:choose statement using drag and drop:

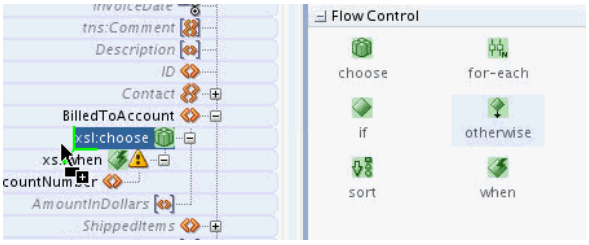
1. In the Components window, select the XSLT Elements page.
2. Expand the Flow Control section. You can click the plus sign (+) next to Flow Control to expand the section.
3. Drag the **choose** icon to the right side of the target node until you can see the green highlighting, as shown in Figure 40-18.

Figure 40-18 Dragging the choose Icon to the Target Node



4. Drop the **choose** icon while the green highlighting is visible. An `xsl:choose` node is added as the parent node of the target node. The `xsl:choose` node contains a child `xsl:when` node.
5. To create the otherwise clause, drag the **otherwise** icon from the Components Window to the left of the xsl:choose node until you can see the green highlighting, as shown in Figure 40-19.

Figure 40-19 Dragging the otherwise Icon to the xsl:choose Node



6. Drop the otherwise icon while the green highlighting is visible. An `xsl:otherwise` node is added as the child node of the `xsl:choose` node.
7. Map the `xsl:when` node to the source node whose existence is to be tested. In our current example, you drag a line from the **HQAccount** node in the source to the **xsl:when** node in the target.

8. Map the xsl:when and xsl:otherwise cases. In our current example, you drag a line from the **HQAccount/AccountNumber** node to the **xsl:choose/xsl:when/AccountNumber** node. Similarly, you drag a line from the **BranchAccount/AccountNumber** node to the **xsl:choose/xsl:otherwise/AccountNumber** node.

   Figure 40-17 shows the completed `xsl:choose` construct.

When viewed in source view, the `xsl:choose` statement looks similar to the following:

```
<BilledToAccount>
   <xsl:choose>
     <xsl:when test="/ns0:PurchaseOrder/HQAccount">
       <AccountNumber>
         <xsl:value-of select="/ns0:PurchaseOrder/HQAccount/AccountNumber"/>
       </AccountNumber>
     </xsl:when>
     <xsl:otherwise>
       <AccountNumber>
         <xsl:value-of select="/ns0:PurchaseOrder/BranchAccount/AccountNumber"/>
       </AccountNumber>
     </xsl:otherwise>
   </xsl:choose>
</BilledToAccount>
```

How to Add Loops Using xsl:for-each

The `xsl:for-each` statement can be used to loop over a source node-set, or set of nodes, and to create output nodes for each node in the source node-set.

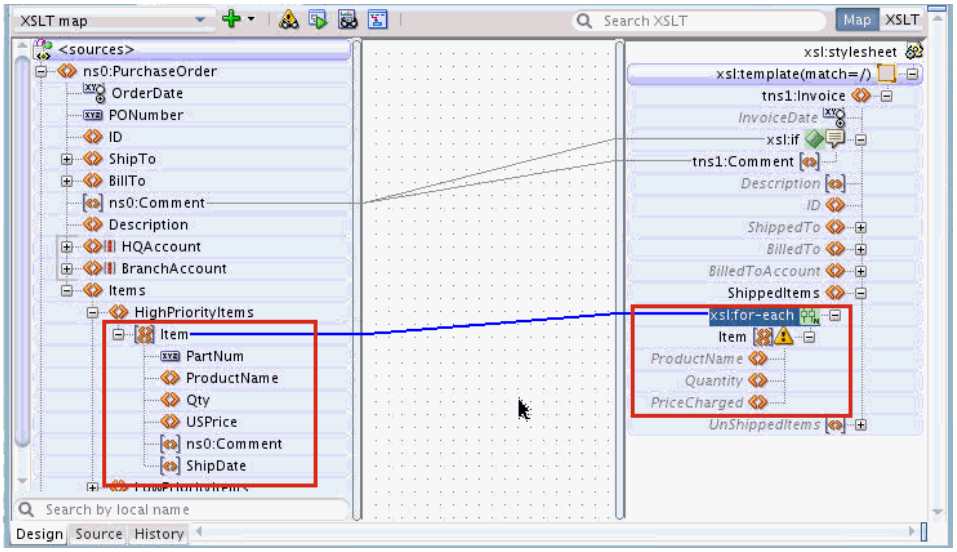
To add an xsl:for-each statement using the context menu:

1. Right-click the target node and select **Add XSL Instruction -> for-each** from the context menu that appears. An `xsl:for-each` statement is added as the parent node of the target node.

2. To set the source node-set to loop over, drag and drop the source node to the `xsl:for-each` statement.

   Figure 40-20 shows an example of creating the `xsl:for-each` statement. The source PurchaseOrder document contains the Item node. The Item node is a repeating node, as represented by its icon. For each Item node in the source document, an Item node is created in the target document using the `xsl:for-each` statement.

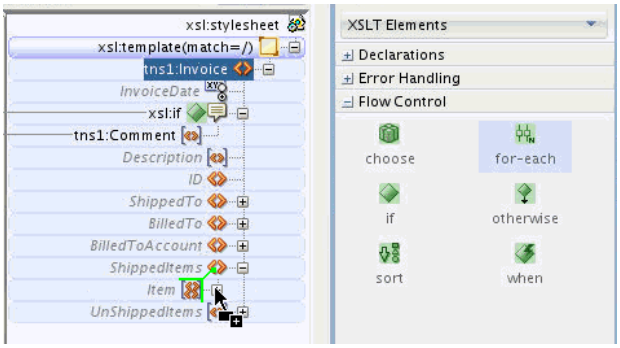
Figure 40-20 Creating an xsl:for-each Statement



To add an xsl:for-each statement using drag and drop:

1. In the Components window, select the XSLT Elements page.
2. Expand the Flow Control section. You can click the plus sign (+) next to Flow Control to expand the section.
3. Drag the **for-each** icon to the right side of the target node until you can see the green highlighting, as shown in Figure 40-21.

Figure 40-21 Dragging the for-each Icon to the Target Node



4. Drop the **for-each** icon while the green highlighting is visible. An `xsl:for-each` node is added as the parent node of the target node.
5. To set the source node-set to loop over, drag and drop the source node to the `xsl:for-each` statement, as shown in Figure 40-20.

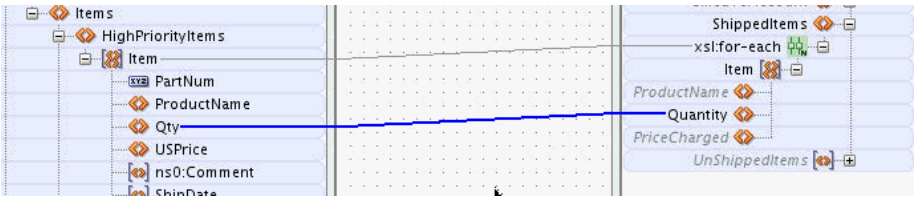
When viewed in the source view, the xsl:for-each statement looks similar to the following:

```
<ShippedItems>
  <xsl:for-each select="/ns0:PurchaseOrder/Items/HighPriorityItems/Item">
     <Item/>
  </xsl:for-each>
</ShippedItems>
```

Note that the Item node, created inside the xsl:for-each statement, is an empty node. You can map elements under the target Item node to set values for them.

For example, as shown in Figure 40-22, if you drag and drop Qty to Quantity, the value of the Qty element is copied to the Quantity element in the output.

Figure 40-22 Mapping Qty to Quantity



The following example shows the resulting code in source view. It also shows a sample source document and output document snippet.

```
<ShippedItems>
  <xsl:for-each select="/ns0:PurchaseOrder/Items/HighPriorityItems/Item">
    <Item>
      <Quantity>
        <xsl:value-of select="Qty"/>
      </Quantity>
    </Item>
  </xsl:for-each>
</ShippedItems>
```

The following snippet shows some sample values for the source document:

```
<HighPriorityItems>
     <Item PartNum="000-AA">
          <Qty>20</Qty>
     </Item>
     <Item PartNum="000-AB">
          <Qty>24</Qty>
     </Item>
</HighPriorityItems>
```

The following snippet shows the output values corresponding to the preceding source document:

```
<ShippedItems>
     <Item>
          <Quantity>20</Quantity>
     </Item>
     <Item>
          <Quantity>24</Quantity>
     </Item>
</ShippedItems>
```

Within an `xsl:for-each` statement, XPath expressions are usually relative to the node selected by the `xsl:for-each` statement. For instance, in the preceding example `Qty` is relative to the current Item node `/ns0:PurchaseOrder/Items/HighPriorityItems/Item`:

```
  <xsl:for-each select="/ns0:PurchaseOrder/Items/HighPriorityItems/Item">
    <Item>
      <Quantity>
        <xsl:value-of select="Qty"/>
      </Quantity>
```

```
    </Item>
  </xsl:for-each>
```

Using absolute paths within the xsl:for-each statement can result in unintended results. For example, if were to use absolute path in the preceding example instead of relative path, the code looks as follows:

```
<ShippedItems>
 <xsl:for-each select="/ns0:PurchaseOrder/Items/HighPriorityItems/Item">
   <Item>
     <Quantity>
       <xsl:value-of
              select="/ns0:PurchaseOrder/Items/HighPriorityItems/Item/Qty"/>
     </Quantity>
   </Item>
 </xsl:for-each>
</ShippedItems>
```

The resultant output document looks like the following:

```
<ShippedItems>
  <Item>
    <Quantity>20</Quantity>
  </Item>
  <Item>
    <Quantity>20</Quantity>       <!-- repeating incorrect value! -->
  </Item>
</ShippedItems>
```

The absolute path always selects the first `Qty` element in the `Item` node-set and you see a repeating value placed into each output `Item` element.

The XSLT Map Editor creates relative paths when mapping nodes under a for-each statement, if possible. It is recommended that you create the `xsl:for-each` statement before mapping the nodes that appear under the for-each. If you map nodes such as `Quantity` before adding the for-each, the editor shows a warning and attempts to refactor the absolute XPath expressions to relative path expressions when you map the node-set to the for-each.

Note:

- Executing an auto map automatically inserts the `xsl:for-each` statement, where required.
- Ensure that your design does not include infinite loops. Infinite loops can result in errors similar to the following during deployment and invocation of your application:

  ```
  ORAMED-04001:
  . . .
  oracle.tip.mediator.service.BaseActionHandler requestProcess
  SEVERE:
  failed reference BPELProcess1.bpelprocess1_client operation =  process
  ```

How to Add xsl:sort for an xsl:for-each Statement

The `xsl:sort` statement can be added to an xsl:for-each statement to specify a field based on which sorting is performed. The `xsl:sort` instruction causes `xsl:for-each` to loop over the defined node-set in a particular order.

To add an xsl:sort statement using the context menu:

1. Right-click the **xsl:for-each** node and select **Add XSL Instruction -> sort** from the context menu that appears.

   The Set Attributes dialog appears.

2. Optionally specify attributes for the `xsl:sort` statement. Click **OK.**

   The Set Attributes dialog enables you to set attributes for the `xsl:sort` statement. Attributes control the way in which the sort is executed. For example, if you select the **'order' Attribute,** you can then select **ascending** or **descending** for the sort order. Select the attributes desired for the sort.

   Note:

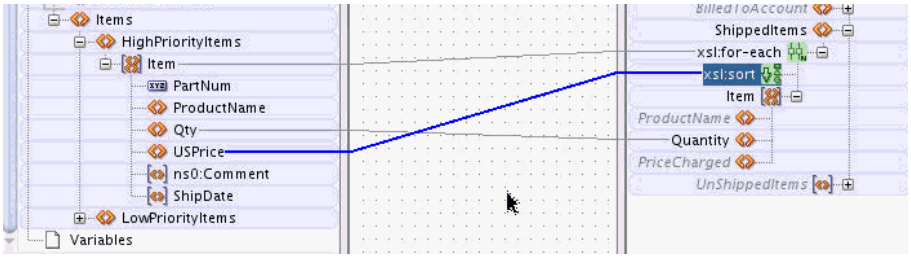
   The default values for attributes are pre-selected in the Set Attributes dialog. These values are used in the absence of any selected attribute.

   For instance, the default for sort order is ascending. You do not have to explicitly select **'order' Attribute** to turn on ascending order.

   The `xsl:sort` statement is added just below the `xsl:for-each` statement and before any other nodes under the for-each.

3. To set the element to sort with, drag and drop a node from under the source node-set to the **xsl:sort** node. For instance, to sort on **USPrice,** drag the **USPrice** node from under the node-set element **Item** to the **xsl:sort** node, as shown in Figure 40-23.

Figure 40-23 Connecting the Source Node-Set to the xsl:sort Node



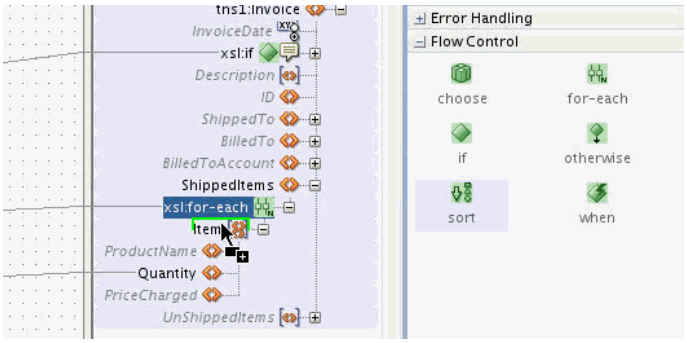
To add an xsl:sort statement using drag and drop:

1. In the Components window, select the XSLT Elements page.

2. Expand the Flow Control section. You can click the plus sign (+) next to Flow Control to expand the section.

3. Drag the **sort** icon to the top of the element below the **for-each** node until you can see the green highlighting, as shown in Figure 40-24.

Figure 40-24 Adding sort to for-each



4. Drop the **sort** icon while the green highlighting is visible. An `xsl:sort` node is added as the sibling node of the highlighted node. In the example shown in Figure 40-24, the `xsl:sort` node is added as a sibling of the Item element.

5. To set the element to sort with, drag and drop a node from under the source node-set to the **xsl:sort** node. For instance, to sort on **USPrice,** drag the **USPrice** node from under the node-set element **Item** to the **xsl:sort** node, as shown in Figure 40-23.

When viewed in the source view, the `xsl:sort` statement looks similar to the following:

```
<xsl:for-each select="/ns0:PurchaseOrder/Items/HighPriorityItems/Item">
  <xsl:sort select="USPrice"/>
  <Item>
    <Quantity>
      <xsl:value-of select="Qty"/>
    </Quantity>
  </Item>
</xsl:for-each>
```

How to Duplicate XSLT Instructions

Sometimes, you must duplicate XSLT instructions in the target tree. For example, you may create two for-each statements next to one another to loop over two node-sets in the source document, or possibly to loop over the same node-set twice.

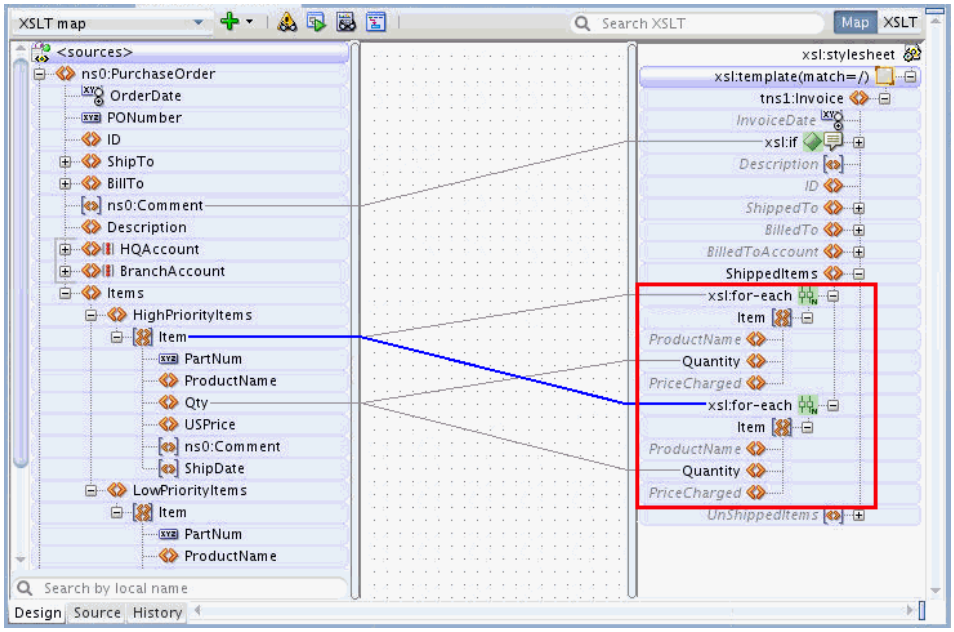
Other XSLT instructions, such as `xsl:if` and `xsl:sort`, can also be duplicated. This section illustrates creating duplicate instructions using the `xsl:for-each` statement. The same process applies to other XSLT instructions.

To duplicate an xsl:for-each statement:

1. Right-click the **xsl:for-each** node in the target tree and select **Duplicate** from the context menu that appears.

   The node is duplicated together with its children and mappings. Figure 40-25 shows the duplicate nodes.

Figure 40-25 Duplicating the xsl:for-each Statement



2. Optionally modify the mapping for the `xsl:for-each` node or change mappings for nodes below the `xsl:for-each` node.

Example: Modifying the Mapping by Changing the XPath Expression

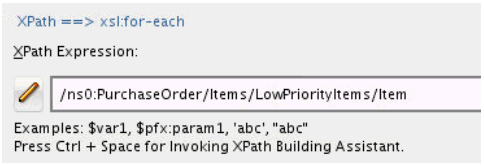
In this example, you modify the duplicate `xsl:for-each` statement to loop over the `Item` nodes under the `LowPriorityItems` node in the source document. There are several ways to modify the mappings. The following example discusses one way to modify the mappings.

If you edit the XPath expression associated with a for-each statement, all relative mappings under the for-each are automatically updated. Use the following steps to modify the XPath expression associated with the duplicate for-each statement:

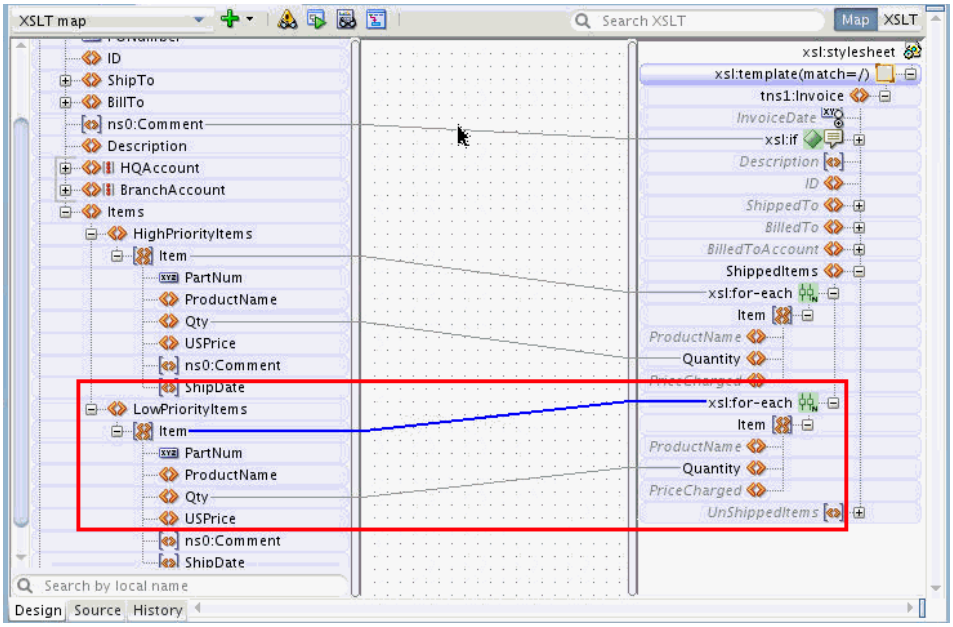
1. To edit the XPath expression, double-click the line connected to the second for-each. This is the blue line in Figure 40-25.

   The Edit XPath dialog appears. The **XPath Expression** field displays the XPath expression corresponding to the map.

2. Change **HighPriorityItems** to **LowPriorityItems** in the XPath Expression. Click **OK.**



   The `xsl:for-each` statement and all its children now reference the `LowPriorityItems/Item` node-set.



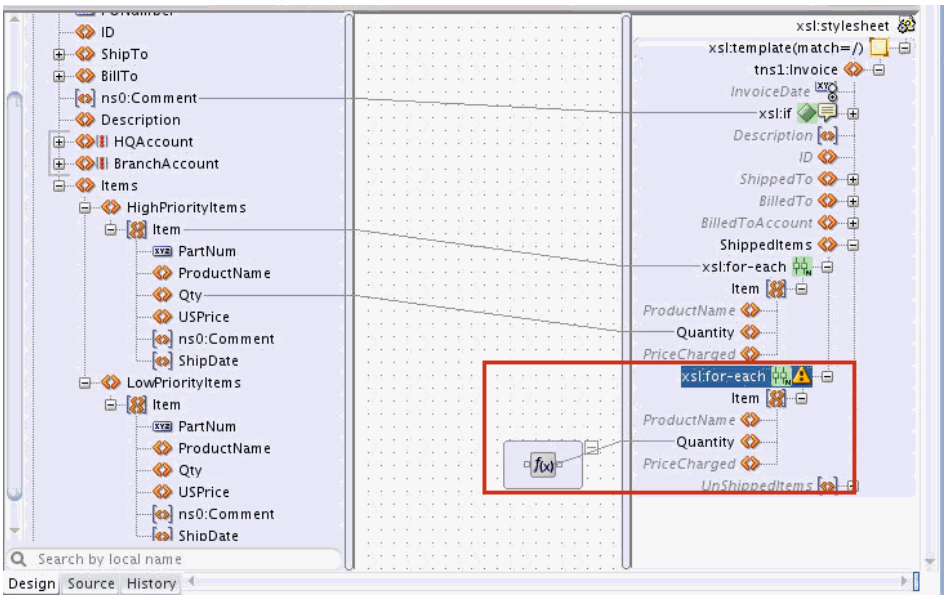
Example: Modifying the Mapping by Deleting and Re-Creating It

To modify the mapping for the for-each statement, you can also choose to remove the mapping and re-create it. The following steps illustrate the process to modify the mappings for the duplicate for-each statement:

1. Right-click the duplicate **xsl:for-each** node and select **Delete Mapping** from the context menu that appears.

   The Refactor XPaths dialog appears asking if you want to refactor the XPath expressions under the for-each statement.

2. Click **No** to refactoring. This keeps the relative paths, as you plan to apply these relative paths to a different loop. After you click No, the lines underneath the for-each statement become temporarily disconnected from the source tree.



In the preceding figure, the relative XPath expression assigned to the `Quantity` field is `Qty`. Without the XPath expression on the for-each statement, the relative path has no `Item` node to be relative to, and consequently, no reference is found in the source tree.

Note:

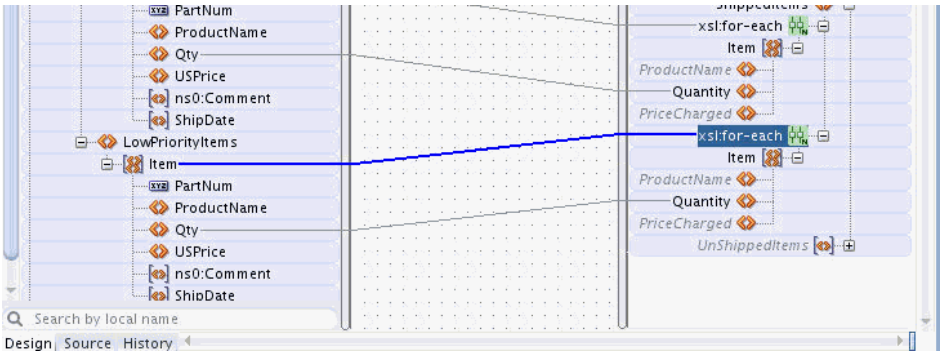
In general, XPath expressions that cannot be resolved to nodes in the source tree are represented in the center panel. This can sometimes indicate an issue, as in the preceding case. There is a relative path that cannot be resolved because of a missing for-each expression above it.

An XPath expression can also be represented in the center pane if the expression is too complex to determine a source reference node at design time.

---

3. Drag and drop a line from the **LowPriorityItems/Item** node in the source tree to the duplicate **xsl:for-each** statement.

   The mapping to the `Quantity` field automatically reconnects to the `Qty` field under the `LowPriorityItems/Item` node, as shown in the following figure.



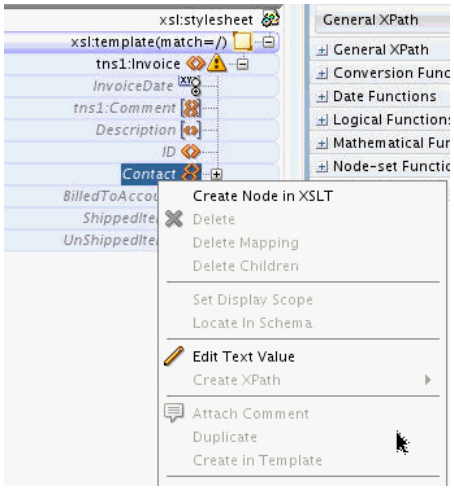
How to Duplicate an Element

In order to duplicate a target node in Map view, the node must have been defined as a repeating node in the target schema. For certain cases, you can create repeating nodes using for-each loops, as discussed in the preceding section. In other cases, you may need to create several independent instances of a node and map data to them through different areas of the source document.

To duplicate a repeating target node:

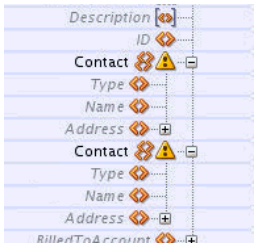
1. If the target node text is grayed, and in italics, right-click the node and select **Create Node in XSLT** from the context menu that appears.

Figure 40-26 Creating Node in XSLT



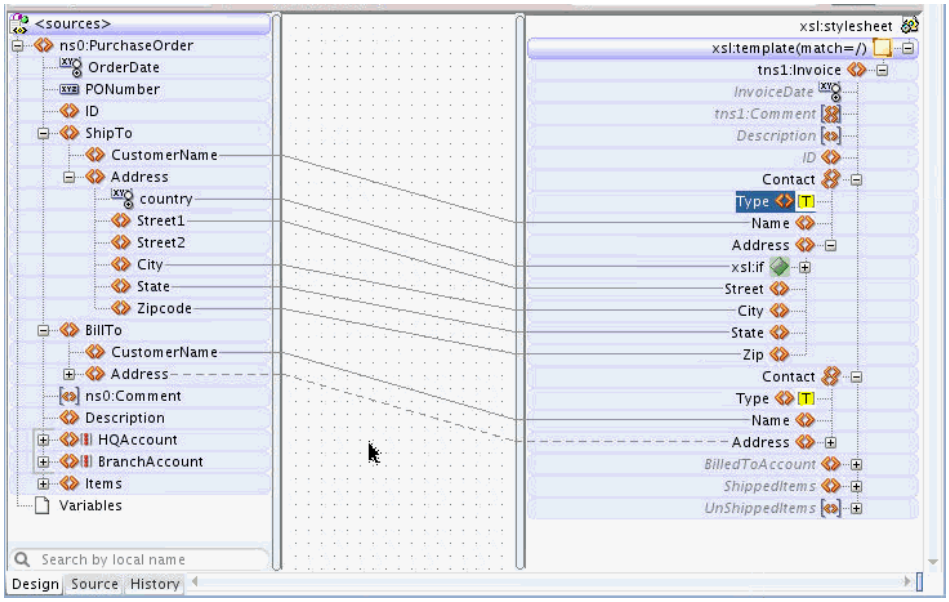
2. Right-click the node again and select **Duplicate** from the context menu that appears. The node is duplicated.

Figure 40-27 Duplicate Contact Nodes



3. Map the appropriate fields from the source document to the two duplicate elements.

   In the following figure, the duplicate `Contact` nodes in the target tree are mapped to different areas of the source document. The first `Contact` node is mapped to the `ShipTo` data. The second Contact node is mapped to the `BillTo` data.



How to Delete an Element or Attribute

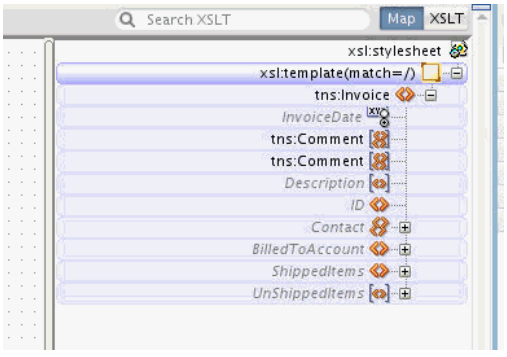
To delete an element or attribute from the current XSLT Map, the element or attribute must first exist in the XSLT. Nodes in the target tree that are not grayed, and not in italics, are nodes that exist in the XSLT. Nodes that are grayed, and in italics, are not part of the XSLT. Such grayed nodes represent candidate elements and attributes from the target schema, and cannot be deleted from the display.

To delete a target node that exists in the XSLT, do one of the following:

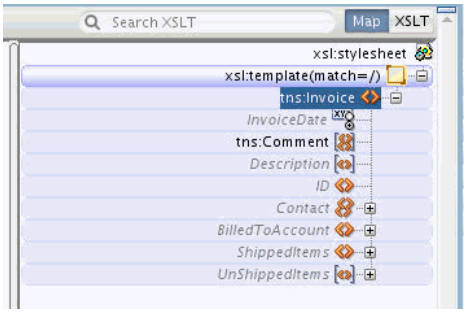
- Click the target node to select it. Press the Delete key.
- Right-click the target node and select **Delete** from the context menu that appears.

The node is removed from the XSLT and any mapping to the node is also removed. The deleted node is not removed from the display. The deleted node becomes gray and italicized indicating that it is now just a possible target node from the target schema, and is no longer part of the XSLT. However, if the deleted node was a duplicate node, or was in a position non-compliant with the target schema, then the node is removed from the display.

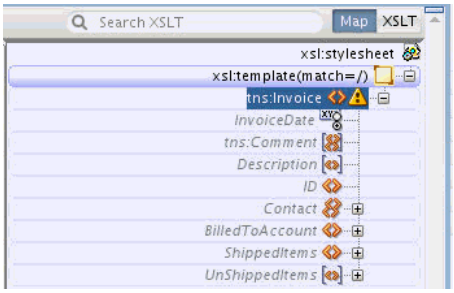
The following figure shows two comment nodes that are part of the XSLT map.



If you right-click the second **Comment** node and select **Delete** from the context menu, the duplicate node is removed from the XSLT and the display, as shown in the figure below.



Next, you right-click and remove the remaining **Comment** node. The node is not removed from the display, but is removed from the XSLT and its appearance changes to gray and italicized, as shown in the following figure.



How to Remove Mappings from an Element or Attribute

To remove the mapping to any target node, right-click the node in the target tree and select **Delete Mapping** from the context menu that appears. Alternatively, you can select the line representing the mapping, and press the Delete key.

When you remove a mapping in Map View, the node that was mapped to is not removed from the XSLT. To remove the node, right-click and select Delete from the context menu that appears. If only the mappings are removed and the node is not, the XSLT generates an empty node when executed.

Editing an XSLT Map in XSLT View

This section discusses basic editing using the XSLT pane in XSLT View. The following list includes the major differences between editing in XSLT View and editing in Map View:

- XSLT View does not provide a merged view of the XSLT nodes and the target schema. The right target pane is divided into two panes. The top pane is called the XSLT pane and the lower pane is called the target pane. If no target schema is defined, then the lower pane is not shown.
- In order to map to target nodes, these nodes must be explicitly added to the XSLT pane before they can be mapped. The editor provides several ways to do this. These are discussed in the subsequent sections.
- The complete range of XSLT 1.0 instructions is available in XSLT View. These instructions can be added anywhere within the XSLT panel, so long as their position is consistent with the XSLT specification.

- A number of advanced features are available in XSLT View, such as template rules (matched templates), named templates, import/include, and so on. These advanced features are discussed later in this chapter.

How to Add a Target Element or Attribute Before Mapping

Before mapping to target elements and attributes, the element or attribute must be explicitly added to the XSLT pane. This section contains the following topics:

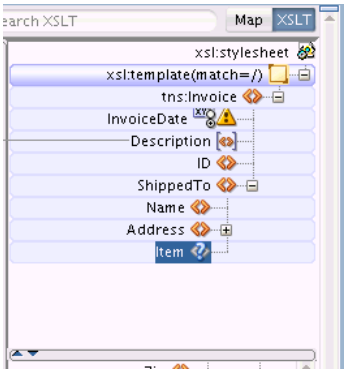
- How to Add Elements and Attributes from the Target Schema
- How to Add Literal Elements and Attributes When No Target Schema Is Present
- How to Create an Empty Node in the Output Document

How to Add Elements and Attributes from the Target Schema

If the target schema is present, you can add elements and attributes from the target tree to the XSLT pane.

You can either use the context menu for an XSLT element or attribute to add a related element/attribute, or drag and drop the desired element/attribute from the target tree to the XSLT tree.

If elements are placed in positions that are inconsistent with the target schema, or if the editor cannot yet determine if the element is valid at the location due to an incomplete XSLT map, then a question mark is shown over the element's icon. The following figure shows an XSLT tree where the Item element is marked with a question mark.



To add elements and attributes when target schema is present:

1. Right-click the element, in the XSLT pane, that is to contain the child element(s) or attribute(s). A context menu appears.

Note:

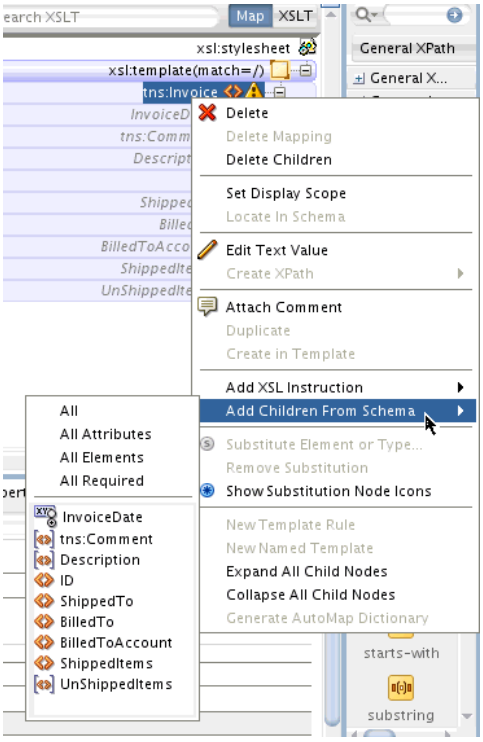
This action can also be executed from Map View if the selected node exists in the XSLT.

In Map View, you can also add an empty XSLT node by choosing **Create Node in XSLT.** See How to Create an Empty Node in the Output Document for more details.

---

2. Select **Add Children From Schema.** A submenu appears with choices that are consistent with the target schema. Figure 40-28 shows the **Add Children From Schema** submenu.

Figure 40-28 Selecting Child Elements/Attributes to Add



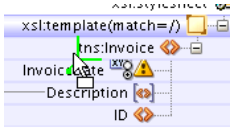
3. Select the element name or attribute name to add. To add all child elements or attributes, select **All Elements** or **All Attributes.** Select **All Required** to add all required child elements or attributes.

To drag and drop elements and attributes from the target schema tree:

- To insert an element as a child of an existing element:

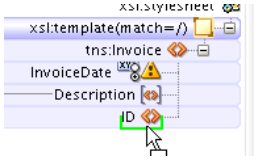
  Drag the target element, or a selected range of elements, from the target pane to the left of the desired element in the XSLT tree. A green highlight appears to indicate that the element is being inserted as a child. Upon drop, the child is appended to the list of children.

  If the green highlight does not appear at a particular position, then it means it is invalid to insert the element at that position.



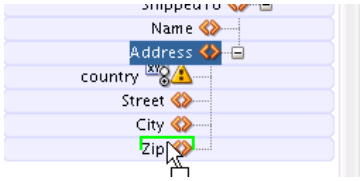
- To insert an element as a sibling after an existing element:

  Drag the target element, or a selected range of elements, from the target pane to the bottom of the existing element in the XSLT tree. A green highlight appears to indicate that the element is being inserted as a sibling. Upon drop, the sibling is appended after the existing element.



- To insert an element as a sibling before an existing element:

  Drag the target element, or a selected range of elements, from the target pane to the top of the existing element in the XSLT tree. A green highlight appears to indicate that the element is being inserted as a sibling. Upon drop, the sibling is appended before the existing element.



- To insert an element as a parent of an existing element:

  Drag the target element, or a selected range of elements, from the target pane to the right of the existing element in the XSLT tree. A green highlight appears to indicate that the element is being inserted as a parent. Upon drop, the element is inserted as the parent of the existing element.

How to Add Literal Elements and Attributes When No Target Schema Is Present

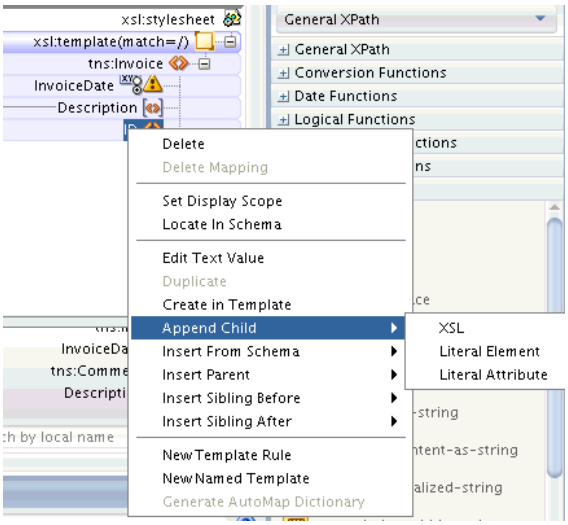
This section applies to both XSLT and Map views.

To add a literal element when there is no target schema:

1. Right-click an element in the XSLT pane. The context menu for the element appears.

2. Select the relative position of the new literal element relative to the existing element. You can create the literal element as the child, sibling, or parent of the existing element. Figure 40-29 shows the available options (**Append Child, Insert Parent, Insert Sibling Before, Insert Sibling After**).

Figure 40-29 Adding a Literal Element to the XSLT Pane



3. Select Literal Element from the submenu. The Define Element dialog appears.
4. Under **Local Name,** enter a name for the literal element. You can optionally specify a namespace for the element.
5. Click **OK** to close the Define Element dialog.

To add a literal attribute when there is no target schema:

1. Right-click an existing literal element in the XSLT pane. The context menu for the literal element appears.
2. Select **Append Child** > **Literal Attribute.** The Define Element dialog appears.
3. Under **Local Name,** enter a name for the literal attribute. You can optionally specify a namespace for the attribute.
4. Click **OK** to close the Define Attribute dialog.

How to Create an Empty Node in the Output Document

When you use the methods discussed under the preceding sections (How to Add Elements and Attributes from the Target Schema and How to Add Literal Elements and Attributes When No Target Schema Is Present) to add nodes to the XSLT pane, these nodes are created as empty nodes unless you map them to source nodes. Also, when you add a complex node to the XSLT pane, all required nodes under the complex node are created automatically.

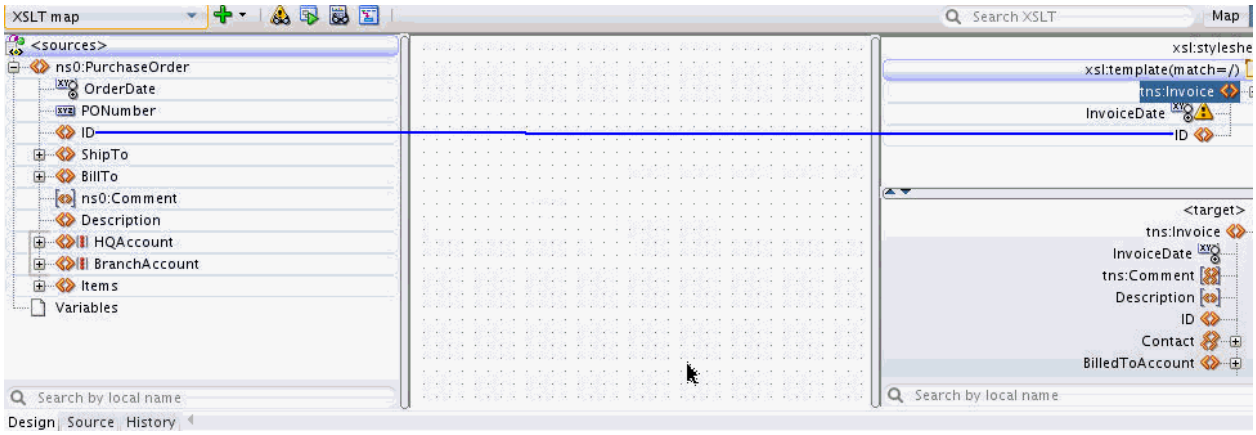
You can also set the XSL Map Initialization Options in the **XSL Maps: XSL Editor preferences** page to generate empty nodes when a map is created. See How to Set the XSL Editor Preferences for more information.

How to Perform a Value Copy by Linking Nodes

To copy the value of an attribute or leaf-element in the source to an attribute or leaf-element in the XSLT pane, drag a line from the source node to the XSLT node. A green highlighted line appears as you are dragging and dropping. When you complete the drop, a line is drawn connecting the source and target nodes.

Figure 40-30 shows the XSLT view where the `PurchaseOrder/ID` source element is mapped to the `Invoice/ID XSLT` element. A line connects the source and XSLT node.

Figure 40-30 Performing a Value Copy



How to Insert an xsl:valueof Statement

When you map a source element to an XSLT element, as described in the preceding section, an implicit xsl:valueof statement is created in the XSLT.

In XSLT View, you can explicitly create multiple `xsl:valueof` elements for an XSLT element. The resultant value of the XSLT element is the concatenation of the individual `xsl:valueof` values.

To create an xsl:valueof element for an XSLT element:

1. Right-click the literal element in the XSLT pane. The context menu appears.
2. Select **Append Child > XSL > value-of** from the context menu. The Set Attributes dialog box appears.
3. Optionally select **'disable output escaping' Attribute.** Click **OK.**

   The `xsl:value-of` element is inserted.
4. Map the `xsl:value-of` element to a source element just as you map a literal element. This is described in the preceding section.

If an `xsl:value-of` element is added to a literal element that has no mapping, the `xsl:value-of` statement appears under the literal element until it is mapped. After you map the `xsl:value-of` element, the editor hides the `xsl:value-of` statement under the literal element and shows only the line indicating the mapping.

If you add multiple `xsl:value-of` statements to the literal element, then all `xsl:value-of` statements are explicitly shown under the literal element, and mapping lines can be separately drawn to each `xsl:value-of` element.

How to Set a Literal Text Value for an XSLT Node

To set a literal text value for an XSLT node:

1. Right-click the node in the XSLT pane. Select **Edit Text Value** from the context menu that appears.

   The Set Text dialog appears.

2. Enter the text value to be assigned to the node. Do not enclose the text in quotation marks.

3. Click **OK.**

   A yellow T icon appears next to the node indicating that a text value has been set for the item. If you move the mouse cursor over the node, the corresponding text appears.

How to Set a Literal Text Value Using an xsl:text Instruction

To set the value for a literal element using the xsl:text instruction:

1. Right-click the literal element in the XSLT pane. The context menu appears.

2. Select **Append Child > XSL > text** from the context menu. The Set Attributes dialog box appears.

3. Optionally select **'disable output escaping' Attribute.** Click **OK.**

   The `xsl:text` element is inserted.

4. Right-click the newly inserted **xsl:text** element and select **Edit Text Value** from the context menu that appears.

   The Set Text dialog box appears.

5. Enter the text value to be assigned Do not enclose the text in quotation marks.

6. Click **OK.**

   A yellow T icon appears next to the `xsl:text` node indicating that a text value has been set for the element. If you move the mouse cursor over the node, the corresponding text appears.

7. To change the text value at any time, right-click the **xsl:text** element again and select **Edit Text Value** from the context menu that appears.

How to Add XSLT Statements

This procedure applies to XSLT View only. XSLT statements can be added using the Components window or context menu.

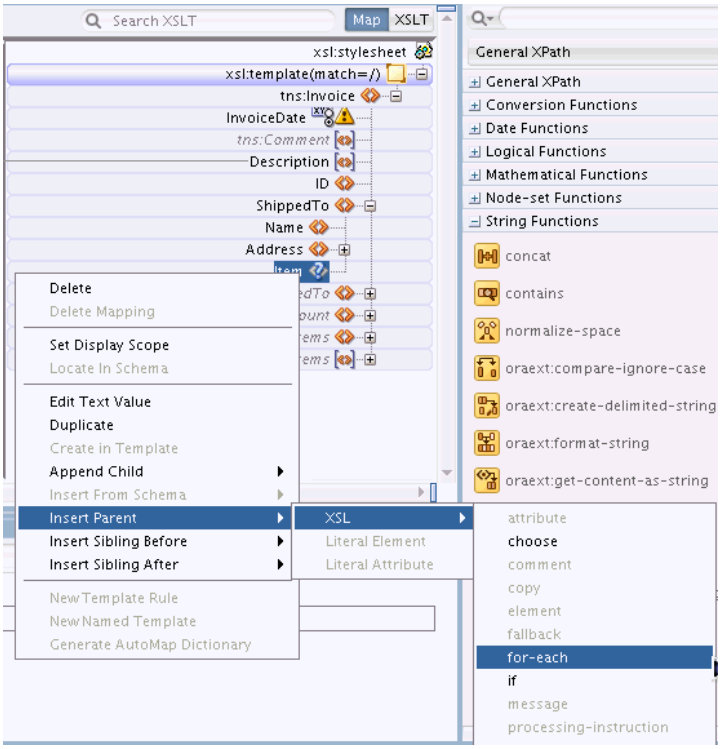
To add an XSLT element using the context menu:

1. Right-click an element in the XSLT pane. The context menu for the element appears.

2. Select the relative position of the new element relative to the existing element. You can create the literal element as the child, sibling, or parent of the existing element. The available options are **Append Child, Insert Parent, Insert Sibling Before,** and **Insert Sibling After.**

3. Select **XSL** from the submenu. A list of available XSLT elements valid for the position is displayed. Figure 40-31 shows a sample XSL selection.

Figure 40-31 Inserting an XSLT Element



4. Select the desired XSLT element.

   Depending on the element selected, a dialog may prompt you for attribute values. If so, then enter the attribute values, and click **OK**.

5. The XSLT element gets added to the tree.

   The attributes of the added element are not explicitly shown in the XSLT tree. Hover your mouse over the element to see its attributes in the tooltip text. Alternatively, select the XSLT element to view and edit the element properties in the Properties window.

To add XSLT elements from the Components window:

1. Make sure that the Components window is visible. The default location is the top right hand corner of Oracle JDeveloper.

2. If the Components window is not visible, select **Components** from the **Window** menu.

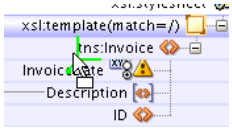
3. Select the **XSLT Elements** page.

4. Select the desired section under **XSLT Elements.** Drag the desired XSLT element to the XSLT pane.

- To insert the XSLT element as a child of an existing element:

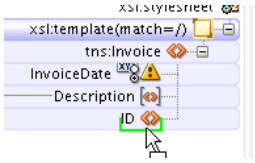
  Drag the XSLT element from the Components window to the left of the existing element in the XSLT tree. A green highlight appears to indicate that the XSLT element is being inserted as a child. Upon drop, the XSLT element is appended to the end of any existing children.

  If the green highlight does not appear at a particular position, then it means it is invalid to insert the element at that position.



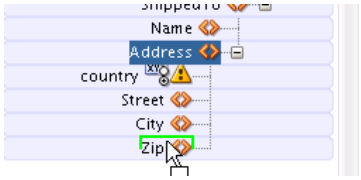
- To insert the XSLT element as a sibling after an existing element:

  Drag the XSLT element from the Components window to the bottom of the existing element in the XSLT tree. A green highlight appears to indicate that the XSLT element is being inserted as a sibling. Upon drop, the XSLT element is appended after the existing element.



- To insert the XSLT element as a sibling before an existing element:

  Drag the XSLT element from the Components window to the top of the existing element in the XSLT tree. A green highlight appears to indicate that the XSLT element is being inserted as a sibling. Upon drop, the XSLT element is appended before the existing element.



- To insert the XSLT element as a parent of an existing element:

  Drag the XSLT element from the Components window to the right of the existing element in the XSLT tree. A green highlight appears to indicate that the XSLT element is being inserted as a parent. Upon drop, the XSLT element is added as the parent of the existing element.

  Depending on the XSLT element selected, a dialog may prompt you for attribute values. If so, then enter the attribute values, and click **OK**.

5. The XSLT element gets added to the tree.

The attributes of the added element are not explicitly shown in the XSLT tree. Hover your mouse over the element to see its attributes in the tooltip text. Alternatively, select the XSLT element to view and edit the element properties in the Properties window.

How to Set the Value of an XSLT Expression Attribute

Many XSLT instructions contain special attributes that are interpreted as XPath expressions. These expression attributes are generally named `select` or `test`. For example, the `xsl:for-each` element contains a `select` attribute and the `xsl:if` element contains the `test` attribute.

Such attributes can be defined by XPath expressions. You can set the values for these attributes using drag and drop to the XSLT element in the XSLT pane (in XSLT View) or target pane (in Map View).

To set the value of an XSLT expression attribute using drag and drop from the source tree, drag a line from the desired node in the source tree to the desired XSLT element in the XSLT pane. A line appears connecting the source tree node to the XSLT element.

The appropriate expression attribute is inserted for the XSLT instruction in the source view. For example:

```
<xsl:if test="/ns0:PurchaseOrder/BillTo/Address/@country">
```

The preceding example code is formed by dragging the `country` attribute in the source schema to the `xsl:if` statement in the XSLT tree. The code causes the `if` condition to test for the presence of the `country` attribute in the source schema.

How to Duplicate an Element

To duplicate a literal element in the XSLT pane that is defined as a repeating node in the target schema, use the instructions under How to Duplicate an Element. The instructions are same as those for the Map View.

If no target schema is defined, you can duplicate any node in XSLT View, except the root node.

If a node needs to be duplicated, but the node is not defined as a repeating node in the target schema, you can create a duplicate node by explicitly creating a literal element as follows:

To duplicate a literal element that is not defined as a repeating node in the target schema:

1. In the XSLT pane, right-click the element to be duplicated. The context menu appears.
2. Select **Insert Sibling After -> Literal Element** from the context menu. The Define Element dialog appears.
3. Enter the element name and namespace of the node to be duplicated. Click **OK.**

How to Delete an Element or Attribute

This feature is available in both the XSLT and Map views. When using Map View, the action is to be performed in the target pane.

To delete a target node that exists in the XSLT, do one of the following:

- Right-click the node in the XSLT pane. Select **Delete** from the context menu that appears.
- Click the node in the XSLT pane to select it. Press the Delete key.

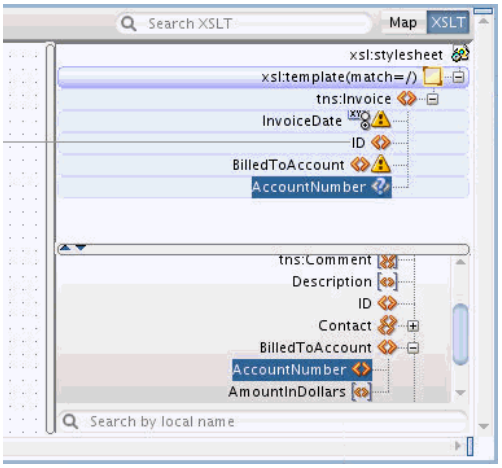
The node is removed from the XSLT pane together with any mappings to the node.

How to Move an Element

You can move an element by dragging it from one position and dropping it to another position in the XSLT pane. The element can be moved to become a sibling, parent, or child of another element. You cannot move an element in Map View.

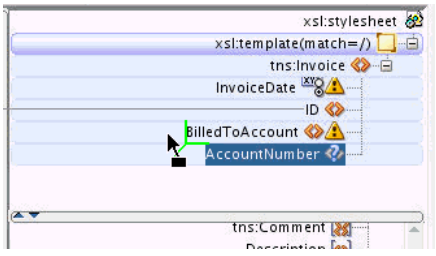
In Figure 40-32, the `AccountNumber` element is at an incorrect location in the XSLT pane. To make it consistent with the target schema, you must move the `AccountNumber` element under the `BilledToAccount` element.

Figure 40-32 AccountNumber Element



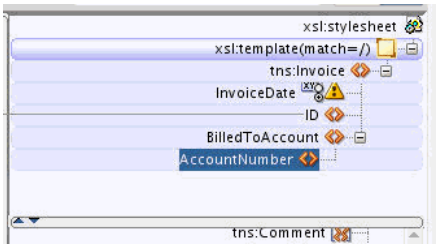
Drag the `AccountNumber` node in the XSLT pane to the left of the `BilledToAccount` node until the green highlight appears, as shown in Figure 40-33. The green highlight indicates that the `AccountNumber` element is dropped as a child of `BilledToAccount.`

Figure 40-33 Dragging the AccountNumber Node



Drop the element while the green highlight is visible. The `AccountNumber` node is repositioned under the `BilledToAccount` node, as shown in Figure 40-34.

Figure 40-34 Repositioned AccountNumber Node



How to Remove Mappings from an Element or Attribute

To remove the mapping to any XSLT node, right-click the node and select Delete Mapping from the context menu. Alternatively, you can select the line representing the mapping, and press the Delete key. You can use the preceding methods in both the Map and XSLT views.

Using XPath Expressions

How to Perform a Value Copy by Linking Nodes and How to Perform a Value Copy by Linking Nodes discussed how to use the drag and drop action to create a mapping between a source and target element or attribute. The drag and drop action creates an XPath expression in the XSLT that references specific nodes in the source document.

For example, the following XSLT code is generated by mapping a source element to a target element:

```
<ID>
 <xsl:value-of select="/ns0:PurchaseOrder/ID"/>
</ID>
```

The preceding code contains an `xsl:value-of` statement. The `select` attribute for this statement contains an XPath expression (/ns0:PurchaseOrder/ID)that references the source node being mapped.

This XPath expression represents a location path expression. XPath expressions can also be complex and include XPath functions and operators.

For example, the following code concatenates the value of the source element `/PurchaseOrder/ID` to the value of the attribute, `/PurchaseOrder/@PONumber.` It then assigns the result to the target element, `<ID>.`

```
<ID>
 <xsl:value-of select="concat(/ns0:PurchaseOrder/ID,/ns0:PurchaseOrder/@PONumber )"/>
</ID>
```

In the preceding code, the value in the `select` attribute is the XPath expression. The XPath expression uses the concat function to concatenate two source node values.

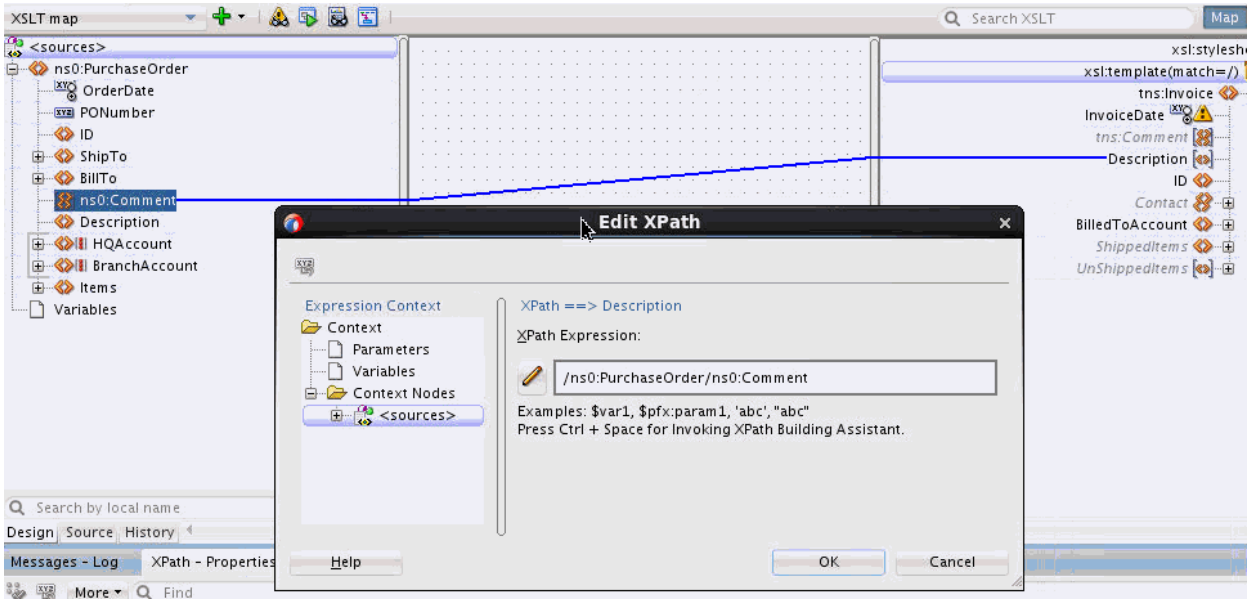
The XSLT Map Editor provides a number of ways to enter more complex XPath expressions than those that are created by simple drag and drop actions. The following methods for creating XPath expressions are available in both Map and XSLT View.

How to Modify an Existing Source to Target Mapping

You can modify the XPath expression for a mapping created from a drag and drop action between a source and target node.

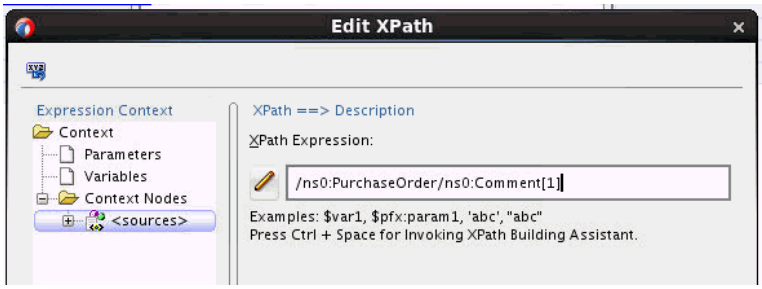
To edit an XPath expression using the Edit XPath dialog

1. Double-click the line representing the source to target mapping. The Edit XPath dialog appears.



2. Edit the **XPath Expression,** as needed.

   For example, if you want the **Description** field to contain the first **Comment** that occurs in the source, you add a predicate to the expression with the index of the first **Comment.**



   Click **Help** if you need more information on editing the XPath expression.

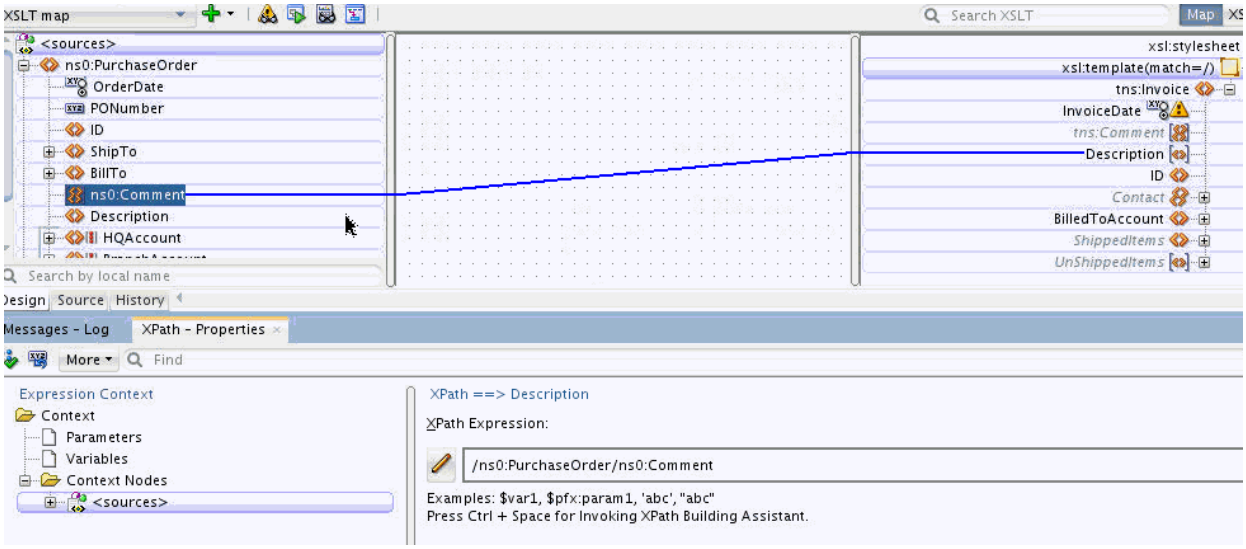
3. Click **OK** in the Edit XPath dialog.

To edit an existing XPath expression using the Properties window

1. If the Properties window is not visible, select **Window > Properties** from the Oracle JDeveloper menu bar.

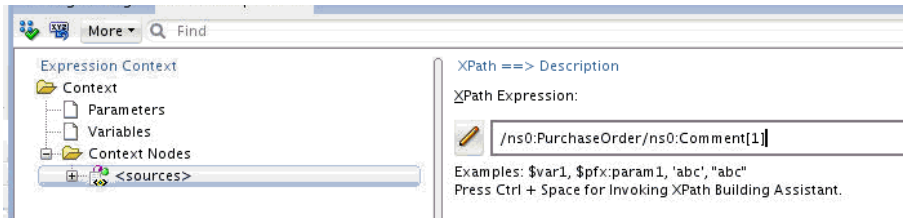
   The default location of the Properties window is below the XSLT Map Editor.

2. Click to select the line representing the source to target mapping. The Properties window shows the XPath expression corresponding to the selected map line.



3. Edit the **XPath Expression,** as needed.

   For example, if you want the **Description** field to contain the first **Comment** that occurs in the source, you add a predicate to the expression with the index of the first **Comment.**



   Click the **Help** icon in the Properties window, if you need more information on editing the XPath expression.

4. To update the XSLT with the changes, click the **Apply Changes** icon in the upper left corner of the Properties window. Alternatively, click anywhere in the XSLT Map Editor.

How to Add an XPath Function to an Existing XPath Expression

You can drag and drop a function onto an existing source to target mapping. When you drop a function on a map, the existing location path expression is used to populate the first parameter of the function that is dropped.

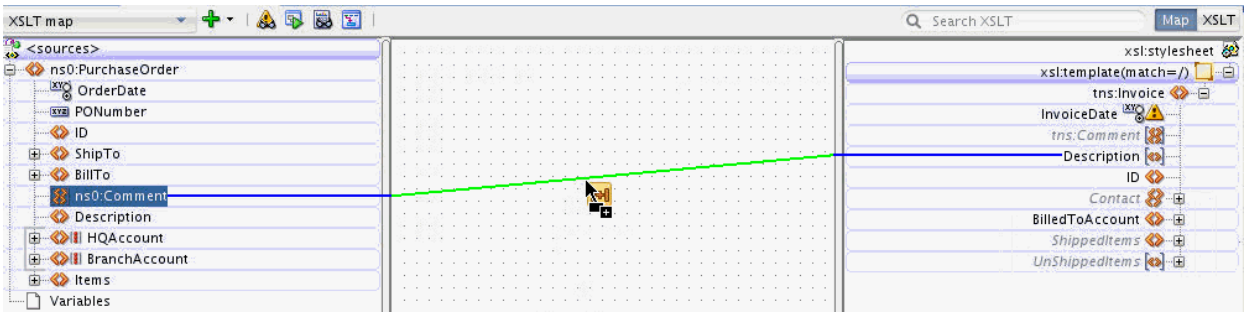
In the following steps, you change the expression you edited in the preceding section (How to Modify an Existing Source to Target Mapping). You use the concat function to concatenate the first **Comment** in the source with the **Description** in the source.

To add an XPath function to an existing XPath expression

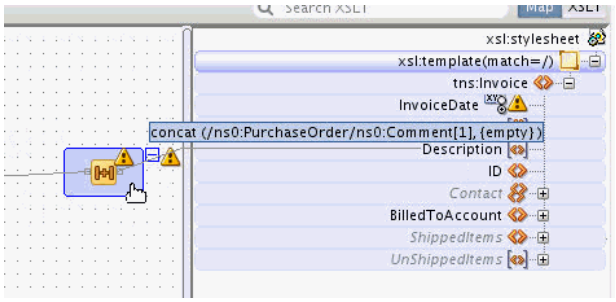
1. If the Components window is not visible, select **Window > Components** from the Oracle JDeveloper menu bar.

2. In the Components window, select **General XPath.** Expand the String Functions section by clicking the plus sign (+) next to it.

3. Drag the **concat** function icon from the String Functions section to the line representing the existing map that you want to modify. The line turns green, indicating that you can drop the function.



4. Drop the **concat** function on the line. The function is inserted into the map, and the first parameter of the concat function is set to the value of the existing XPath expression.



Note:

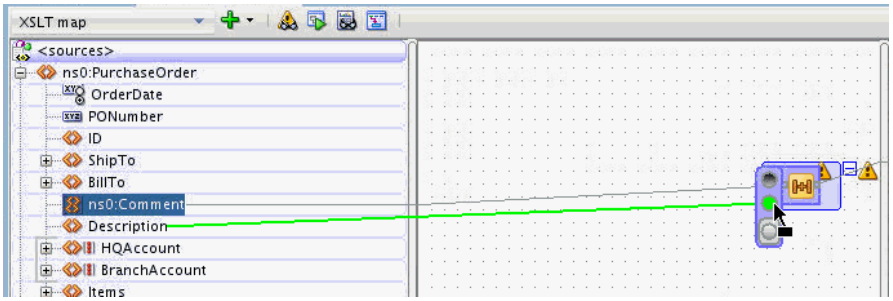
If a function does not get added to the map, the function may not have any parameters. For example, if you drag and drop the **xp20:current-date** function onto the existing line, it has no effect because the `xp20:current-date` function takes no parameters.

How to Modify an Existing Function XPath Expression in the Canvas Pane

XPath functions are shown in the canvas panel and can be edited in several ways. Continuing our example from the previous section (How to Add an XPath Function to an Existing XPath Expression), you set the value of the second parameter of the concat function in several ways.

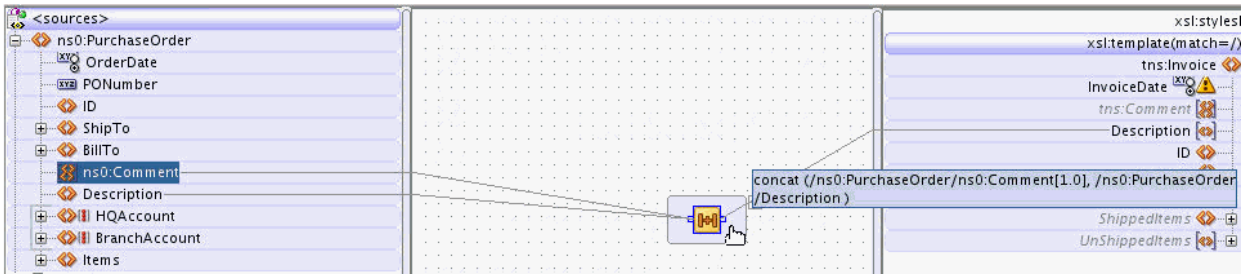
To set a function parameter using drag and drop:

1. Drag a line from the **Description** element in the source tree to the left side of the **concat** function icon in the canvas pane. A pop-up panel appears with connectors for each possible parameter in the function.



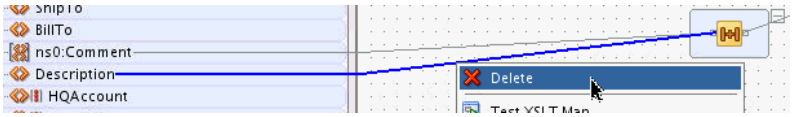
2. Drop the line on the desired connector. In the preceding figure, you drop the line on the second connector, which represents the second required parameter. You can also choose to drop the line on the third optional parameter, and fill in the second parameter value later.

   The parameter is added to the function. The warning icon disappears after all required parameters have been added.



To delete a function parameter:

To delete a function parameter, select the line representing the input to the function parameter and press the Delete key. Alternatively, you can right-click the line and select **Delete** from the context menu.

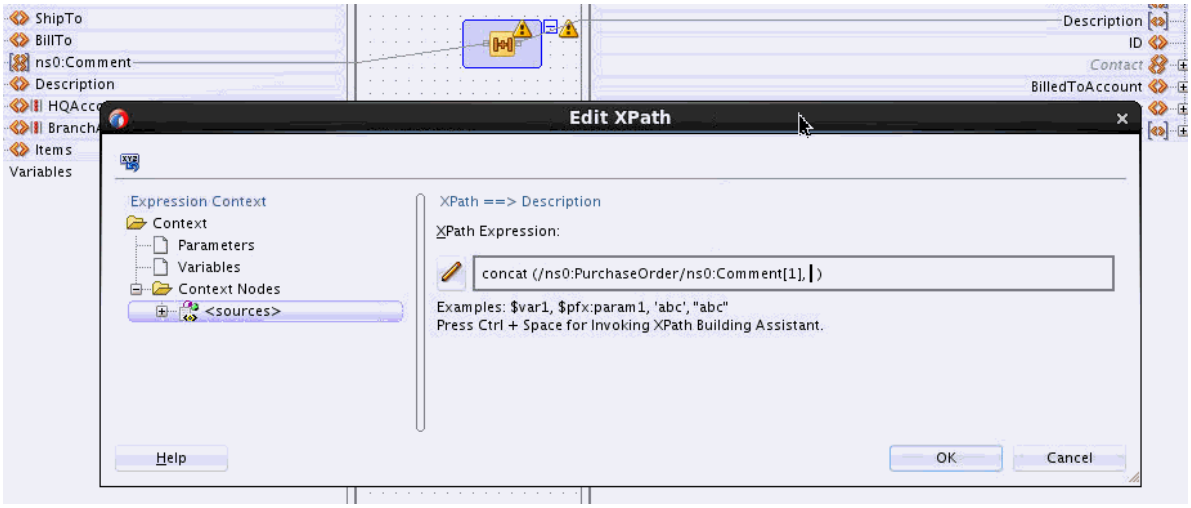


How to Edit a Function as a Full XPath Expression

You can edit an XPath function as a textual XPath Expression using the XPath Edit dialog or the XPath Edit panel in the Properties Window.

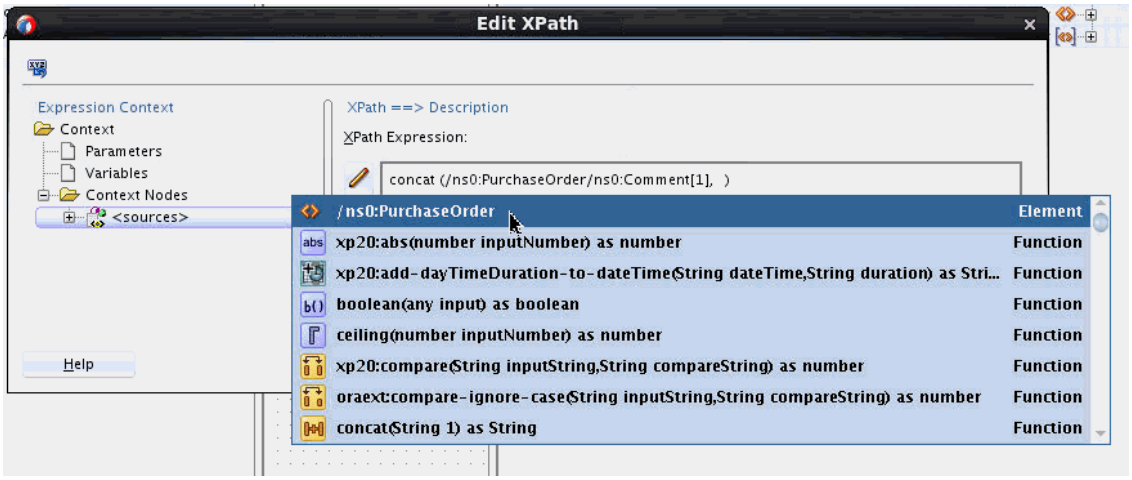
To edit a function as a textual XPath expression using the XPath Edit dialog:

1. Double-click the expression folder in the canvas pane, in the area bordering the function icon. The Edit XPath dialog appears.

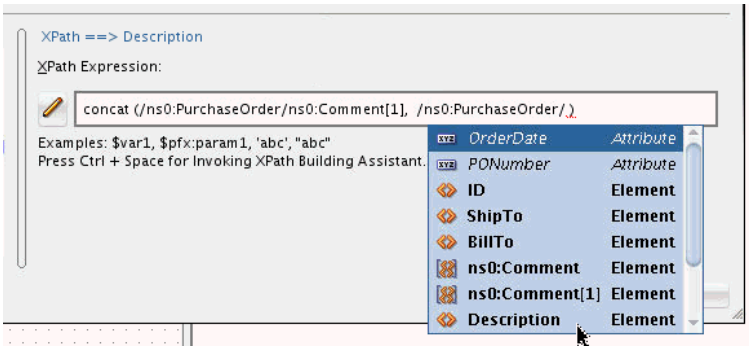


If you double-click the center icon instead, it brings up the Edit Function dialog.

2. Edit the **XPath Expression,** as desired. To add the XPath location path for the Description node, for example, place the cursor after the first parameter. Press **Ctrl + Space** and double-click **/ns0:PurchaseOrder** to select it.



`/ns0:PurchaseOrder` is inserted in the expression and the drop-down menu is populated with the possible children of the `/ns0:PurchaseOrder` node.



3. Double-click the **Description** entry to select it. You can also put the mouse cursor on an entry, and press the Enter key to select it.

4. Click **OK** in the Edit XPath dialog.

To edit a function as a textual XPath expression using the Properties Window:

1. If the Properties window is not visible, select **Window > Properties** from the Oracle JDeveloper menu bar.

   The default location of the Properties window is below the XSLT Map Editor.

2. Click the expression folder in the canvas pane, in the area bordering the function icon. The full **XPath Expression** for the function appears in the right pane of the Properties window.

3. Edit the XPath Expression, as desired. You can also refer to Steps 2 to 3 in the preceding procedure.

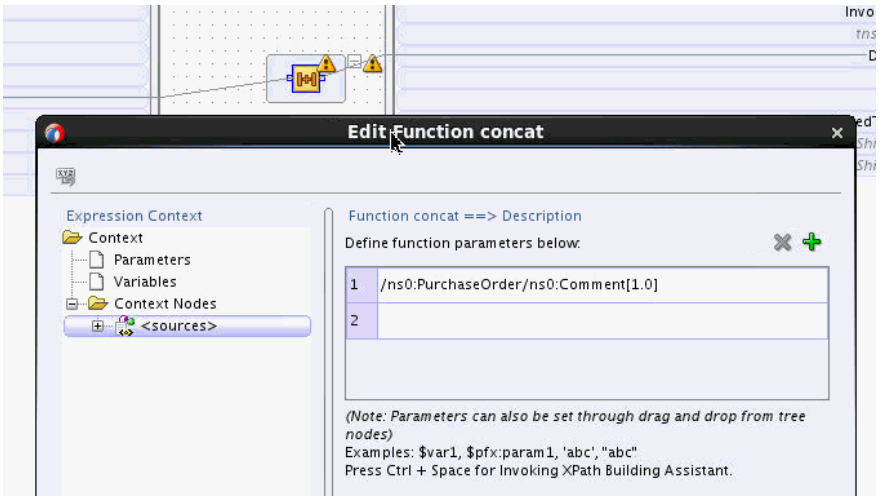
4. To update the XSLT with the changes, click the **Apply Changes** icon in the upper left corner of the Properties window. Alternatively, click anywhere in the XSLT Map Editor.

How to Edit Individual Function Parameters

The XSLT Map Editor can parse a function into its corresponding parameters, so that the XPath for each parameter can be edited in a separate XPath Expression field.

To edit the parameters of a function using the Edit Function dialog:

1. Double-click the function icon in the canvas pane. The Edit Function dialog appears.



   Make sure you double-click the center function icon. Double-clicking the area bordering the function icon brings up the Edit XPath dialog.

2. Edit the function parameters individually, as desired. Optionally click **Help** for more information about editing the parameters.

3. Click the **Add** icon, represented by the green plus sign (+), to optionally add a new parameter.

4. Click **OK** after you finish editing the parameters.

To edit the parameters of a function using the Properties window:

1. If the Properties window is not visible, select **Window > Properties** from the Oracle JDeveloper menu bar.

   The default location of the Properties window is below the XSLT Map Editor.

2. Click the center area of the function icon in the canvas pane. The function parameters appear in the right pane of the Properties window.

3. Edit the function parameters, as desired. Optionally click the **Help** icon for more information about editing the parameters.

4. Click the **Add** icon, represented by the green plus sign (+), to optionally add a new parameter.

5. To update the XSLT with the changes, click the **Apply Changes** icon in the upper left corner of the Properties window. Alternatively, click anywhere in the XSLT Map Editor.

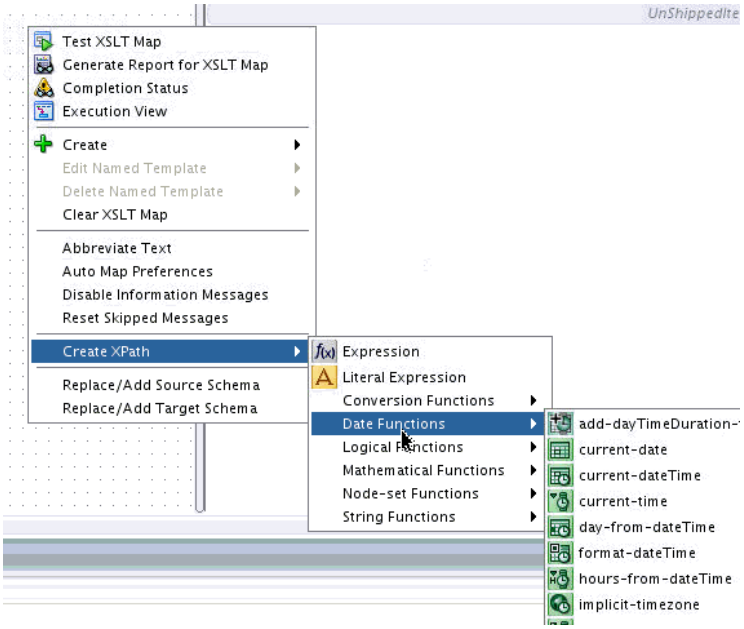
How to Create a New Function in the Canvas Pane

There are several ways to create a new function in the XSLT canvas pane. These are described in the sections that follow.

To create an XPath Function using the canvas context menu

1. Right-click a blank area in the center canvas pane. Select **Create XPath** from the context menu that appears.

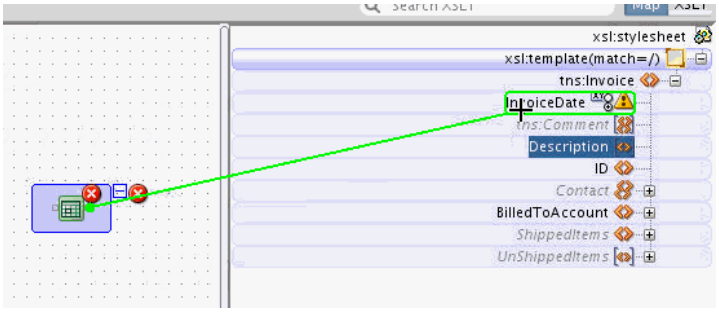
2. Select the desired function from the **Create XPath** submenu. For example, select the **current-date** function from the **Date Functions** category.



An Information dialog may appear, prompting you to connect the function to a target node. Click **OK.**

The function icon appears on the canvas pane.

3. Map the function to a target node by dragging a line from the function to the target node.



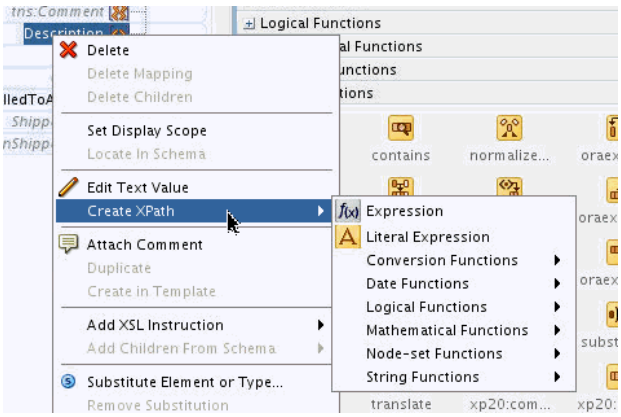
4. If the function requires parameters, edit the parameters using one of the methods discussed in How to Modify an Existing Function XPath Expression in the Canvas Pane.

To create an XPath function using the Components window

1. If the Components window is not visible, select **Window > Components** from the Oracle JDeveloper menu bar.
2. In the Components window, select **General XPath** or **Advanced XPath.** Select a category of functions, for example, **String Functions.**
3. Drag the desired function from the Components window to the center canvas pane of the XSLT Map Editor.
4. Map the function to a target node by dragging a line from the function to the target node.
5. If the function requires parameters, edit the parameters using one of the methods discussed in How to Modify an Existing Function XPath Expression in the Canvas Pane.

To create an XPath function using the target tree context menu

1. Right-click the target tree node (Map View) or the XSLT tree node (XSLT View) to which the XPath function needs to be assigned. The context menu appears.
2. Select **Create XPath.** Select the desired XPath function from the submenu that appears.

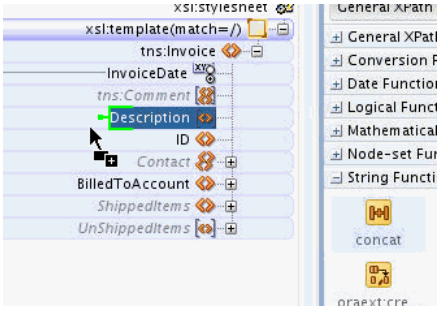


The function is created in the canvas pane and linked to the target/XSLT node for which it was created.

3. If the function requires parameters, edit the parameters using one of the methods discussed in How to Modify an Existing Function XPath Expression in the Canvas Pane.

To create an XPath function by dragging it to the target tree

1. If the Components window is not visible, select **Window > Components** from the Oracle JDeveloper menu bar.
2. In the Components window, select **General XPath** or **Advanced XPath.** Select a category of functions, for example, **String Functions.**
3. Drag the desired function from the Components window to the target tree node (Map View), or XSLT tree node (XSLT View), to which the function is to be assigned. A green highlight appears to the left of the target/XSLT tree node.



4. Drop the function while the green highlight is visible.

   The function is created in the canvas pane and linked to the target/XSLT node where the function was dropped.

5. If the function requires parameters, edit the parameters using one of the methods discussed in How to Modify an Existing Function XPath Expression in the Canvas Pane.

How to Chain Functions Together

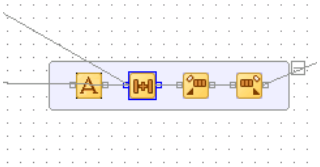
To chain one function to another:

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 40-35 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 40-35 Chaining Functions



How to Remove an XPath Expression

To remove an XPath Expression:

1. Select the XPath expression/function icon in the Canvas pane.
2. Right-click the icon and select **Delete** from the context menu.

How to Import 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 Map Editor 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. To download these and other samples, see the Oracle SOA Suite samples.

   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`

   The text for these functions is as follows:

```
// SampleExtensionFunctions.java
package oracle.sample;
/*
This is a sample XSLT Map Editor 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 Creating User-Defined XPath Extension Functions for more information on the format of this file. The following syntax represents the functions `toKilograms` and `replaceChar` as they are coded in Step 1.

```
<?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.Sam pleExtensionFunctions`.
   - 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 40-1 can be used for parameter and return values:

Table 40-1 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 Components window under the **User Defined** page in the **User Defined Extension Functions** group.

7. To make the functions available in the runtime environment, see How to Deploy User-Defined Functions to Runtime for details.

Using Auto Map to Map Complex Nodes

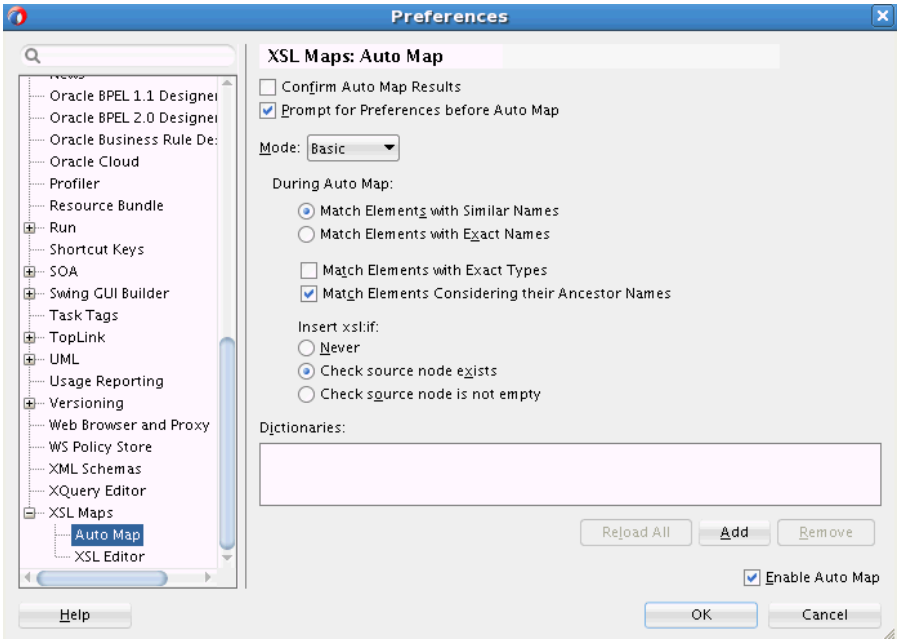
When you map a non-leaf source element to a non-leaf target element, the Auto Map feature assists you by automatically matching the child source elements to their corresponding target elements. Auto Map looks at the element names, types, and paths to come up with the correct mappings. Auto map can also insert `xsl:if` statements for optional nodes, depending on your preferences.

The Auto Map feature is available only when a target schema is used. You can use Auto Map in both Map View and XSLT View.

How to Set Auto Map Preferences

You can specify the behavior of the Auto Map feature using the Preferences dialog. Select **Preferences** from the **Tool** menu. In the navigation tree on the left, select **Auto Map** under **XSL Maps.** Figure 40-36 shows the default settings for the Auto Map preferences.

Figure 40-36 Auto Map Preferences



The following list describes the various Auto Map Preference settings that you can configure:

- **Confirm Auto Map Results:** If you select this option, Auto Map displays a list of matching source and target elements prior to automatically mapping these elements. You can choose the matches that you'd like to be applied.
- **Prompt for Preferences before Auto Map:** If you select this option, the Auto Map Preferences dialog appears every time you try to map two complex nodes.
- **Mode:** Determines whether the Auto Map executes in Basic or Advanced mode. The mode selection determines the rest of the options that appear in this dialog.

The following are the rest of the options available when Basic Mode is selected:

- **Match Elements with Similar Names:** Elements with similar names are matched.
- **Match Elements with Exact Names:** Elements with exactly same names are matched.
- **Match Elements with Exact Types:** Only elements with exactly same data types are matched.
- **Match Elements Considering Their Ancestor Names:** Element path is considered along with the element name when matching.
- **Insert xsl:if:** Determines if xsl:if statements are automatically inserted. The following settings are used:

  - **Never:** xsl:if statements are not inserted automatically.
  - **Check source node exists:** An xsl:if statement is inserted to check for the existence of the source node before the node is created in the output.
  - **Check source node is not empty:** An xsl:if statement is inserted to check that the source node is not empty before creating the node in the output.

The following are the rest of the options available when Advanced Mode is selected:

- **Ancestor Weight:** A number between 0 and 5 indicating the emphasis to be placed on matching of ancestors. The number 0 corresponds to turning the Match Elements Considering Their Ancestor Names option off in Basic mode. The number 5 corresponds to turning the **Match Elements Considering Their Ancestor Names** option on in Basic mode.
- **Linguistic Weight:** A number between 0 and 5 indicating the emphasis to be placed on matching of element names. The number 0 indicates that the element names need not match. The number 5 indicates that the element names must be an exact match.
- **Type Weight:** A number between 0 and 5 indicating the emphasis to be placed on matching of element names. The number 0 indicates that the element types need not match. The number 5 indicates that the element types must be an exact match.
- **Match Threshold (%):** The Auto Map computes a percentage match for each map (Ancestor, Linguistic, Type), and selects the highest percentage amongst these. If the highest match is above the threshold percentage, then a match is made.
- **Dictionaries:** Enables you to add existing dictionaries to the Auto Map. Dictionaries can be defined from existing maps and used in subsequent maps.

How to Execute an Auto Map

To execute an Auto Map:

1. Drag and drop a complex source node to the target element in the XSLT pane. If you are using Map View, then you'd drop the source node to a node in the target pane.
2. Depending on your Auto Map Preferences, the Preferences dialog might appear. Select your Auto Map preferences, and click **OK.**
3. Depending on your Auto Map Preferences, the Auto Map dialog might appear. Verify the matches created by the Auto Map, and click **OK.**

Checking the Completion Status of the Map

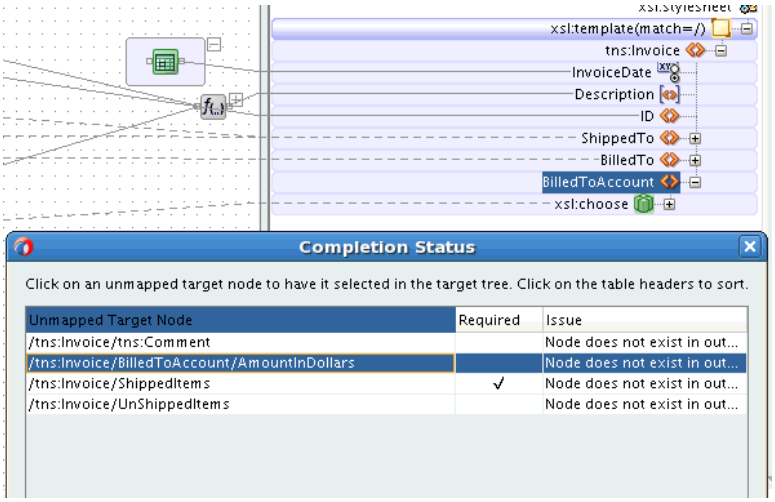
If you are using a target schema for your map, you can check the completion status of the map at any time. You can do this in both Map View and XSLT View. The completion status check flags the following:

- All unmapped target elements and attributes. A flag indicates if the target element is a required element in the target schema.
- Target elements mapped with incomplete XPath expressions. For instance, an XPath function, mapped to a target node, might be missing a parameter.

- All missing target elements and attributes. A flag indicates if the missing target element is a required element in the target schema.

To check the completion status of a map, right-click the Canvas (center) pane, and select **Completion Status** from the context menu. The Completion Status dialog appears showing all incomplete target nodes. Clicking a row in the Completion Dialog status selects the corresponding node location in the XSLT/target tree. Figure 40-37 shows the Completion Status dialog with a missing node highlighted.

Figure 40-37 Completion Status Dialog

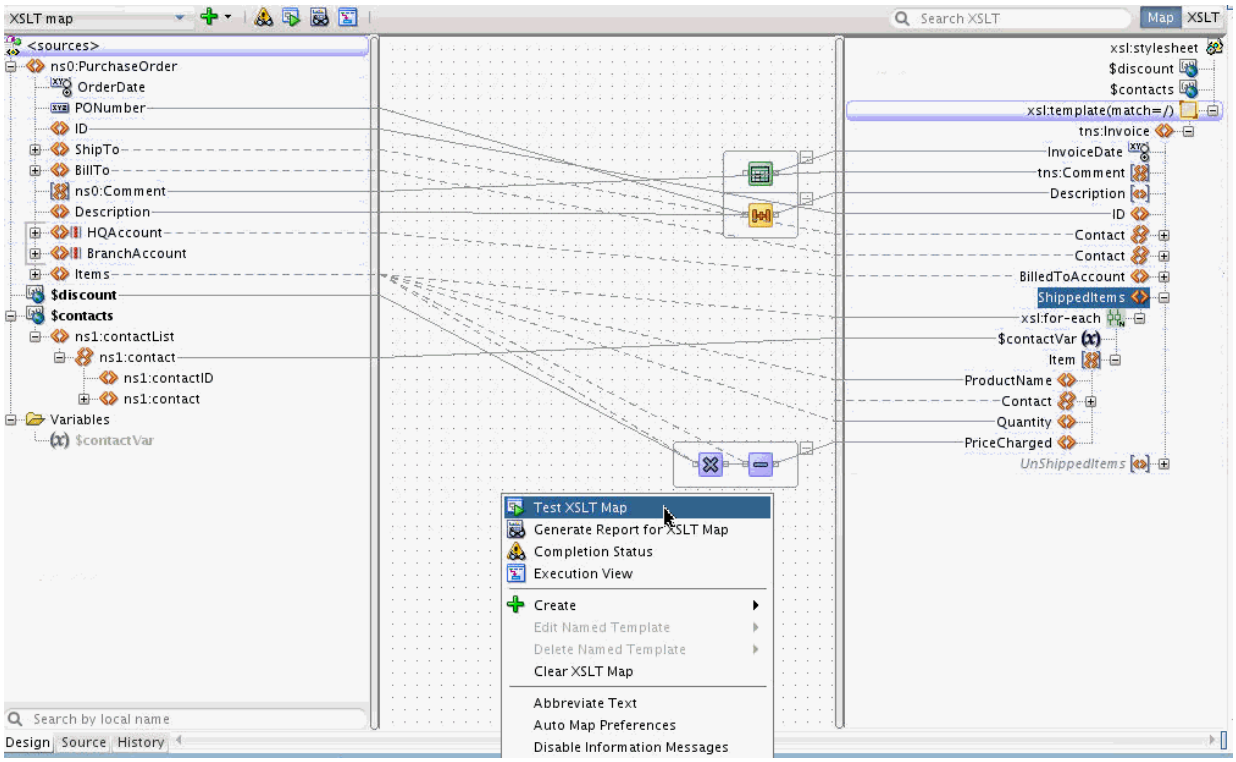


Testing the Map

The XSLT Map Editor provides a tool to test the map. To invoke the test tool, right-click the Canvas pane, and select **Test** from the context menu. You can use the test tool in both Map View and XSLT View.

Figure 40-38 demonstrates launching the Test XSL Map dialog.

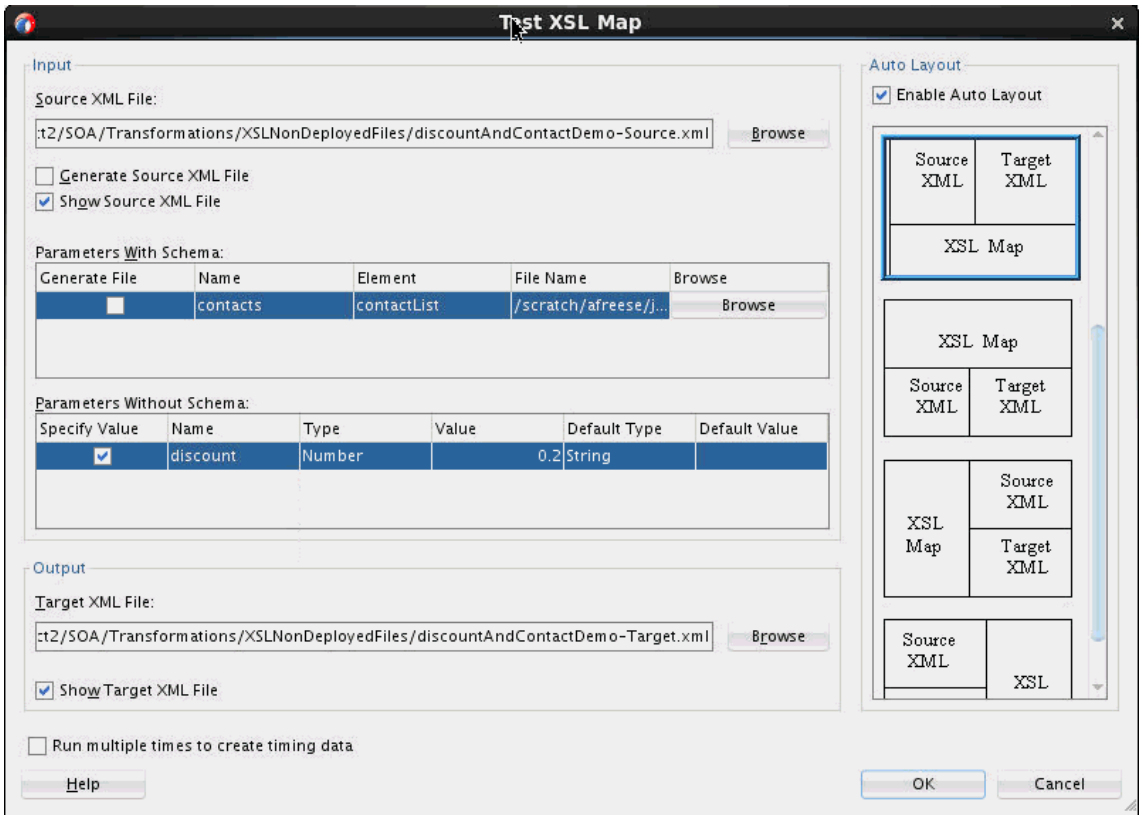
Figure 40-38 Invoking the Test Dialog



How to Test the Transformation Mapping Logic

The Test XSL Map dialog shown in Figure 40-39 enables you to test the transformation mapping logic you designed with the XSLT Map Editor. 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 40-39 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** check box to create a sample XML file based on the map source XSD schema.

3. Select the **Show Source XML File** check box to display the source XML files for the test. The source XML files display in an Oracle JDeveloper XML editor.

   If the map has defined parameters, the **Parameters With Schema** or **Parameters Without Schema** 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** check box to generate a file.

   b. If the **Parameters Without Schema** table appears, you can specify a value by selecting the **Specify Value** check box and making appropriate edits to the **Type** and **Value** columns.

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** check box to display the target XML file for the test. The target XML file displays in an Oracle JDeveloper XML editor.

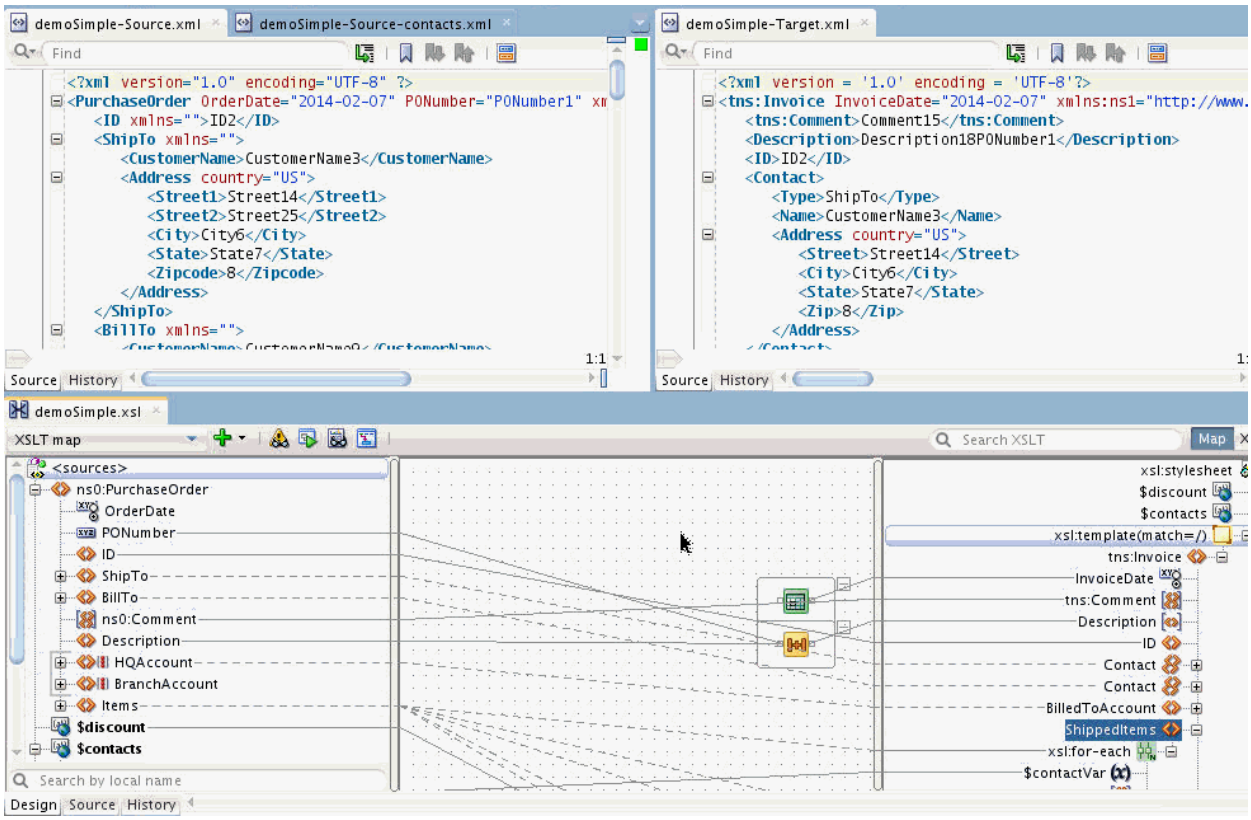
6. If you select to show both the source and target XML, you can customize the layout of your XML editors. Select **Enable Auto Layout** in the upper right corner and click one of the patterns.

7. Click **OK**.

   The test results shown in Figure 40-40 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 40-40 Test Results



How to Test XSLT Maps that Use DVM Lookup Functions

You can test an XSLT map that contains DVM lookup functions. If your map uses DVM lookup functions that reference local files or files in the MDS, and these files are accessible from your JDeveloper environment, then you need not perform any additional steps.

If your map uses DVM lookup functions that reference files not accessible in your JDeveloper environment, then you can create local DVM files for testing without requiring to modify the DVM references in your XSLT. Use the following steps:

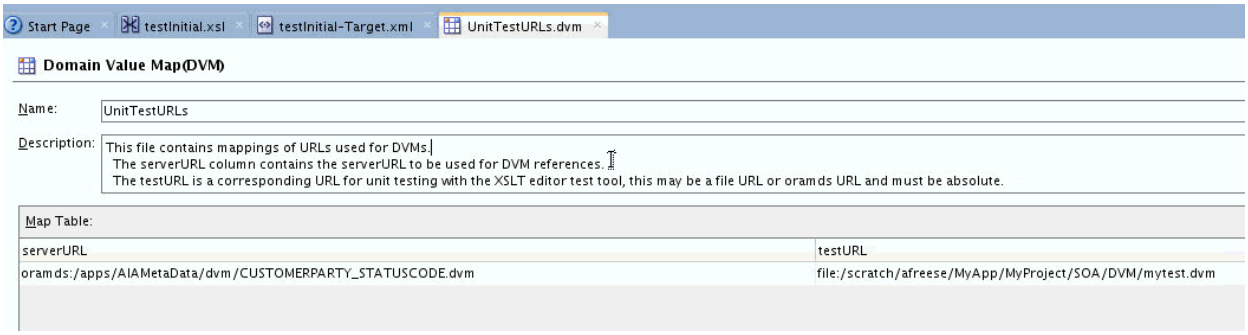
1. If you have not already run the XSLT test, execute the test tool once. The test tool generates the file UnitTestURLs.dvm in the XSLNonDeployedFiles folder, located in the same folder as your XSLT file.

   The UnitTestURLs.dvm file contains mappings between the DVM references in your XSLT file and DVM references to local test files. For example, if the XSLT file that you are testing has a reference to the file, `oramds:/apps/AIAMetaData/dvm/CUSTOMERPARTY_STATUSCODE.dvm`, but you do not have access to this file in JDeveloper, then you can create a local DVM lookup file against which the test is performed.

2. Open the UnitTestURLs.dvm file, located in the XSLNonDeployedFiles folder, in JDeveloper.

3. Under the **serverURL** column, add the reference for the DVM file that you reference in your XSLT.

4. Under the **testURL** column, add the reference to a local file to be used for testing.



5. Save the UnitTestURLs.dvm file.

6. Populate your test DVM file with test data.

7. Execute the test tool. The lookup is performed against the local file. You do not need to modify your XSLT to point to the local file. The test uses the UnitTestURLs.dvm file to look up the correct test file for the reference defined in the **serverURL** column.

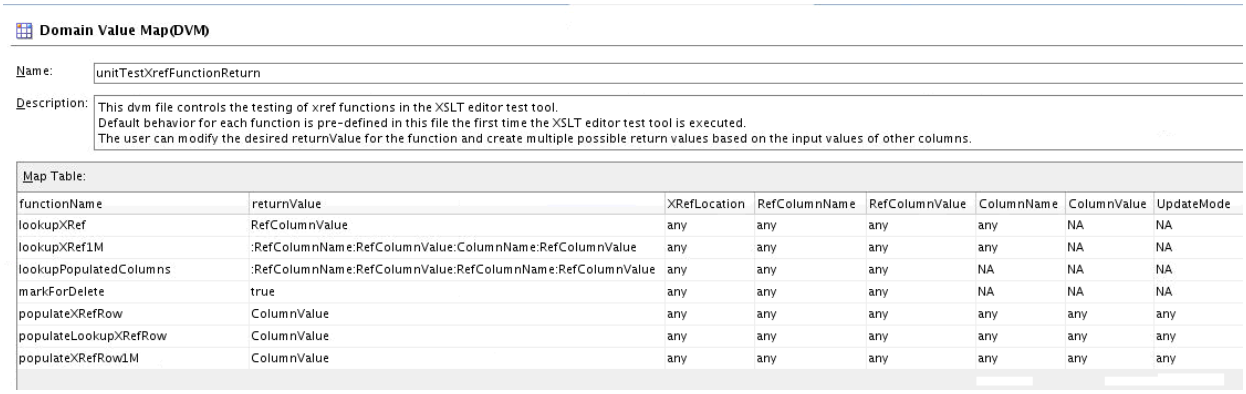
How to Test XSLT Maps that Use XREF Functions

You can use the local dvm lookup file, called UnitTestXrefFunctionReturn.dvm to emulate the test. This file is automatically generated when you run the XSLT test for the first time.

The UnitTestXrefFunctionReturn.dvm file includes default responses for all the XREF functions. This simulates the expected responses when the functions execute correctly. You may modify the default responses. You can also create different return values for different calls of the same function when the parameter values are different.

The following figure shows the initial DVM file.

Figure 40-41 The UnitTestXrefFunctionReturn.dvm File



Domain Value Map(DVM)

Name: unitTestXrefFunctionReturn

Description: This dvm file controls the testing of xref functions in the XSLT editor test tool.
Default behavior for each function is pre-defined in this file the first time the XSLT editor test tool is executed.
The user can modify the desired returnValue for the function and create multiple possible return values based on the input values of other columns.

Map Table:

functionName	returnValue	XRefLocation	RefColumnName	RefColumnValue	ColumnName	ColumnValue	UpdateMode
lookupXRef	RefColumnValue	any	any	any	any	NA	NA
lookupXRef1M	:RefColumnName:RefColumnValue:ColumnName:RefColumnValue	any	any	any	any	NA	NA
lookupPopulatedColumns	:RefColumnName:RefColumnValue:RefColumnName:RefColumnValue	any	any	any	NA	NA	NA
markForDelete	true	any	any	any	NA	NA	NA
populateXRefRow	ColumnValue	any	any	any	any	any	any
populateLookupXRefRow	ColumnValue	any	any	any	any	any	any
populateXRefRow1M	ColumnValue	any	any	any	any	any	any

The **functionName** column specifies the name of the function. To start with, there is only one entry for each function with the default behavior defined. All XREF functions execute with this default information. You can optionally create more entries for a given function, and enter different return values for the function based on the input parameters.

The **returnValue** column specifies the return value from the function. This defines what you would like to see returned from the function.

The other columns define qualifiers that you can use to differentiate one function call from another, based on the value of a given parameter. Each of these columns define a parameter available in a given function call. Not all parameters are available in all functions. When a parameter is not available, it is marked as **NA** (Not Available) in the original table.

Working with returnValue:

The **returnValue** column can either be defined as a text value, such as SBL_001, or may be defined by a parameter name. For instance, if we look at the first function, **lookupXRef,** this function has a return value of **RefColumnValue.** As this is the name of a parameter (**RefColumnValue**), the value of this parameter is the return value of the function to the XSLT.

For example, if the call to the **lookupXRef** function looks like the following:

```
lookupXRef( oramds:/apps/AIAMetaData/xref/CUSTOMERPARTY_PARTYLOCATIONID.xref",
"COMMON_ID", "COMMON_001", "SBL_ID", false())
```

Then the value `COMMON_001` is returned, as this is the value of the **RefColumnValue** parameter that was passed.

For the **markForDelete** function, the value `true` is returned, converted to `Boolean`.

For the **lookupXRef1M** and **lookupPopulatedColumns** functions, a node-set is returned by the function. This node-set contains elements of the following form:

```
<column name="columnNameHere">columnValueHere</column>
```

As shown in Figure 40-41, the default value for the **lookupXRef1M** function is:

```
:RefColumnName:RefColumnValue:ColumnName:RefColumnValue
```

This encodes the column names and values for two column nodes that are returned in a node-set from the function. The first character defines the delimiter to be used in parsing the information. If your data contains a colon (**:**), you can use any character as

the delimiter that is not in your test data, by putting that character as the first character and using it to delimit the data (say, `#abc:def#abc:ghi`).

For example, if we have the following function call to `lookupXRef1M`:

```
lookupXRef1M ("oramds:/mydata", "COMMON_ID", "COMMON_001", "SAP_ID", false())
```

Then using the default definition, for a return value of `:RefColumnName:RefColumnValue:ColumnName:RefColumnValue,` you would receive back two column elements:

```
<column name="COMMON_ID">COMMON_001</column>
<column name="SAP_ID">COMMON_001</column>
```

If you change the line in the DVM to have a `returnValue` of `:SAP_ID:SAP_001:SBL_ID:SBL_001:ORCL_ID:ORCL_001,` then the function returns three column nodes:

```
<column name="SAP_ID">SAP_001</column>
<column name="SBL_ID">SBL_001</column>
<column name="ORCL_ID">ORCL_001</column>
```

Adding Additional Rows:

You can also add additional rows to the DVM file. You can add additional rows for a function by providing different input values for the parameters resulting in different return values.

For example, if we have several lookups against the same XREF file, but want to get different values back from each lookup, we could add the following lines for `lookupXRef` to the DVM file:

Map Table:

functionName	returnValue	XRefLocation	RefColumnName	RefColumnValue	ColumnName	Column'	Upd
lookupXRef	RefColumnValu	any	any	any	any	NA	NA
lookupXRef1M	:RefColumnNai	any	any	any	any	NA	NA
lookupPopulatedColumns	:RefColumnNai	any	any	any	NA	NA	NA
markForDelete	true	any	any	any	NA	NA	NA
populateXRefRow	ColumnValue	any	any	any	any	any	any
populateLookupXRefRow	ColumnValue	any	any	any	any	any	any
populateXRefRow1M	ColumnValue	any	any	any	any	any	any
lookupXRef	SBL_001	oramds:/apps/AIAMetaData/xref/CUSTOMERPARTY_PARTYLOCATIONID.xref	COMMON_ID	COMMON_001	SBL_ID	NA	NA
lookupXRef	SAP_001	oramds:/apps/AIAMetaData/xref/CUSTOMERPARTY_PARTYLOCATIONID.xref	COMMON_ID	COMMON_001	SAP_ID	NA	NA
lookupXRef	ORCL_001	oramds:/apps/AIAMetaData/xref/CUSTOMERPARTY_PARTYLOCATIONID.xref	COMMON_ID	COMMON_001	ORCL_ID	NA	NA

In determining the correct return value, the design time emulator finds the first matching set of parameter values by starting at the bottom of the DVM table. The return value corresponding to the first matching row is returned.

As illustrated in the preceding figure, a call to `lookupXRef("oramds:/apps/AIAMetaData/xref/CUSTOMERPARTY_PARTYLOCATIONID.xref", "COMMON_ID", "COMMON_001", "SBL_ID", false()),` for example, would return the value, `SBL_001.`

A call to `lookupXRef("oramds:/apps/AIAMetaData/xref/CUSTOMERPARTY_PARTYLOCATIONID.xref", "COMMON_ID", "COMMON_002", "SBL_ID", false()),` on the other hand, does not match any of the last three rows, and returns the default value `COMMON_002` (the value of the parameter, `RefColumnValue`).

How to Generate Reports

You can generate an HTML report with the following information:

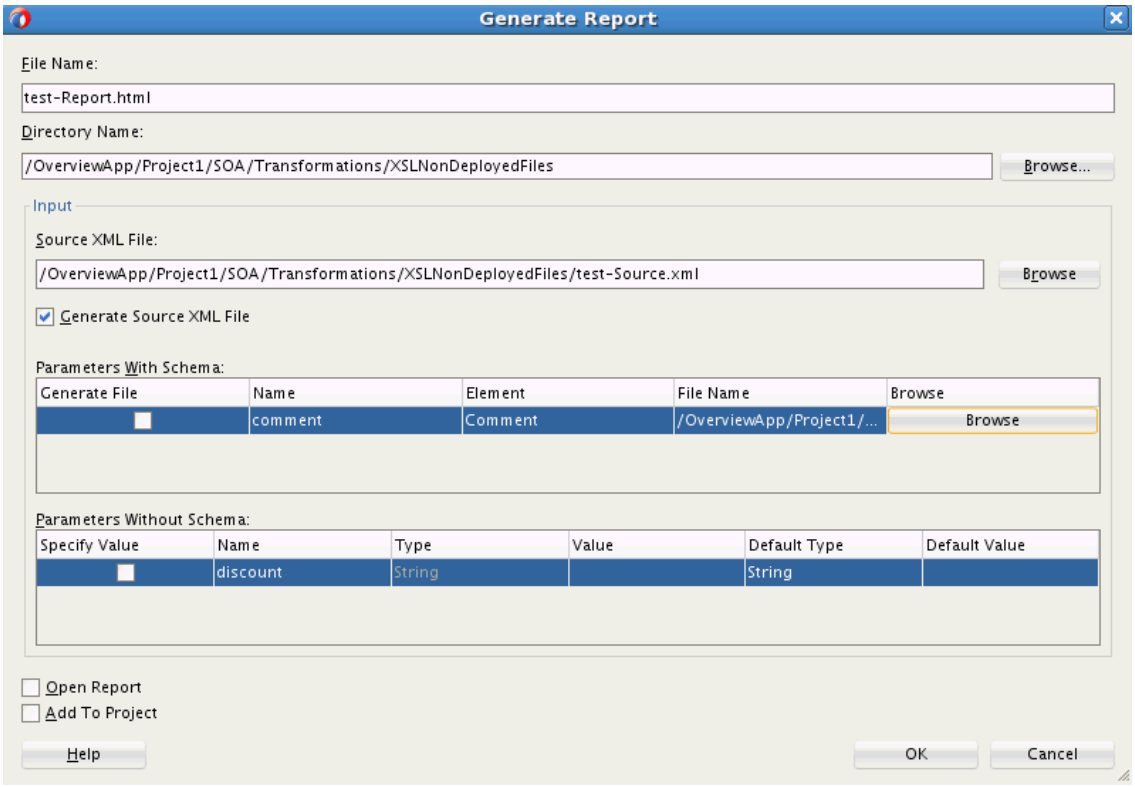
- XSL map file name, source and target schema file names, their root element names, and their root element namespaces
- Target document mappings
- Target fields not mapped (including mandatory fields)
- Sample transformation map execution

Follow these instructions to generate a report.

1. In the Canvas (center) pane, right-click and select **Generate Report for XSLT Map**.

   The Generate Report dialog appears, as shown in Figure 40-42. If the map has defined parameters, the appropriate parameter tables appear.

Figure 40-42 The Generate Report Dialog



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

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. You can modify the following settings under Sample XML Generation:

- 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.

Importing an External XSLT Map

If you have an XSLT map that has been developed with an editor other than JDeveloper, you can import it into JDeveloper.

To import an external map:

1. From the File main menu, select **New > From Gallery.**
2. Under Categories, select **General > XML.** Under Items, select **XSL Map from XSL Stylesheet.** Click **OK.** The XSLT Chooser dialog appears.
3. Select the XSLT file to be imported. Click **OK.** The file is opened and a default header is inserted with no source or target schema definition.
4. To create source and target schema definitions, right-click the Canvas (center) pane, and select **Replace/Add Source Schema** to set the source schema. Select **Replace/Add Target Schema** to set the target schema.

Note:

Imported maps can use Map View only if both the source and target schemas are defined and there are no XSLT features not supported in Map View.

All maps can use the XSLT View.

Using Variables and Parameters

You can add variables and parameters to the XSLT map. These are available in both Map View and XSLT View.

How to Add Global Variables

Global variables can be used in both Map View and XSLT View.

To create a global variable:

1. Right-click any node in the source pane and select **Add Global Variable** from the context menu. Alternatively, click the **Add** icon, identified by the green plus sign in the XSLT toolbar and select **Add Global Variable.** This option is also available on the canvas context menu under the **Create** option.

   The Variable dialog appears.

2. Enter a name for the variable, and an optional namespace and prefix if desired.

3. Click **OK.**

   The variable node appears at the top of the XSLT pane or target pane depending on whether you are using the XSLT View or Map View.

   The variable also appears in the source tree within the Variables folder. This enables you to map from the variable to XPath expressions or nodes in the target tree.

   Note:

   You cannot define a structure for the variable in the current release. If the variable you are referencing represents a complex structure, you can reference nodes within the structure by entering the appropriate XPath expression manually.

How to Add Local Variables in Map View

To add a local variable in Map View:

1. Right-click an existing node in the target tree (not grayed/italicized) and select **Add XSL Instruction > variable** from the context menu that appears.

   The Variable dialog appears.

2. Enter a name for the variable, and an optional namespace and prefix if desired.

3. Click **OK.**

   The variable is added to the target tree, just above the node that you selected.

   The variable also appears in the source tree within the Variables folder. This enables you to map from the variable to XPath expressions or nodes in the target tree.

   To determine if the variable is in scope for a particular XSLT node or XPath expression, select the target tree node or XPath expression. If the variable is in scope for the target tree node or XPath expression, then the variable appears in bold in the source tree. If the variable is not in scope for the selected target tree node or XPath expression, then the variable appears disabled in the source tree.

   Only scalar variables can be defined. You cannot define the structure of a variable. If the variable you are referencing represents a complex structure, you can reference nodes within the structure by entering the appropriate XPath expression manually.

How to Add Local Variables in XSLT View

In XSLT View, local variables are added in the same manner as other XSLT elements. See How to Add XSLT Statements for details about adding XSLT elements using the context menu or Components window.

So, for example, if you select **Insert Sibling Before > XSL > Variable** from the context menu of an XSLT node, you get the Variable dialog box. Enter the name of the variable, optionally specify a namespace, and click **OK.**

The variable appears at the appropriate place in the XSLT/target pane. You can choose to map XPath expressions to the variable to set the value of the variable.

The variable also appears in the source tree under the Variables folder. This enables you to map from the variable to other XPath expressions or XSLT nodes.

To determine if the variable is in scope for a particular XSLT node or XPath expression, select the XSLT node or XPath expression. If the variable is in scope for the XSLT node or XPath expression, then the variable appears in bold in the source tree. If the variable is not in scope for the selected XSLT node or XPath expression, then the variable appears disabled in the source tree.

Only scalar variables can be defined. You cannot define the structure of a variable. If the variable you are referencing represents a complex structure, you can reference nodes within the structure by entering the appropriate XPath expression manually.

Note:

If you are using XSLT 1.0, and using a complex variable, it might be necessary to wrap the variable in the `ora:node-set` function before an XPath expression can be used to access nodes within the variable.

For example, say the `myVar` variable has the following structure:

```
<xsl:variable name="myVar">
  <A>
     <B>sometext</B>
  </A>
</xsl:variable>
```

The text in B can be referenced as `ora:node-set($myVar)/A/B.` The node-set function is not necessary in XSLT 2.0.

How to Add Global Parameters

Parameters can be added to the XSLT map editor both as global parameters and named template parameters.

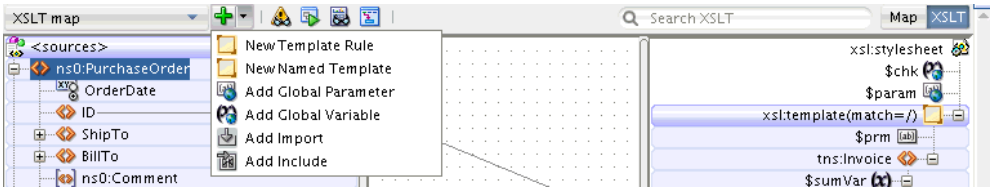
You can add global parameters when creating an XSLT map. See How to Create an XSLT Map for more details. You can also add global parameters to an existing map.

To add a global parameter to an existing map:

1. Right-click any node in the source pane, and select **Add Global Parameter** from the context menu.

Alternatively, click the **Add** icon, identified by the green plus sign, in the XSLT toolbar, and select **Add Global Parameter.** Figure 40-43 shows the XSLT toolbar, which resides at the top of the XSLT Map Editor.

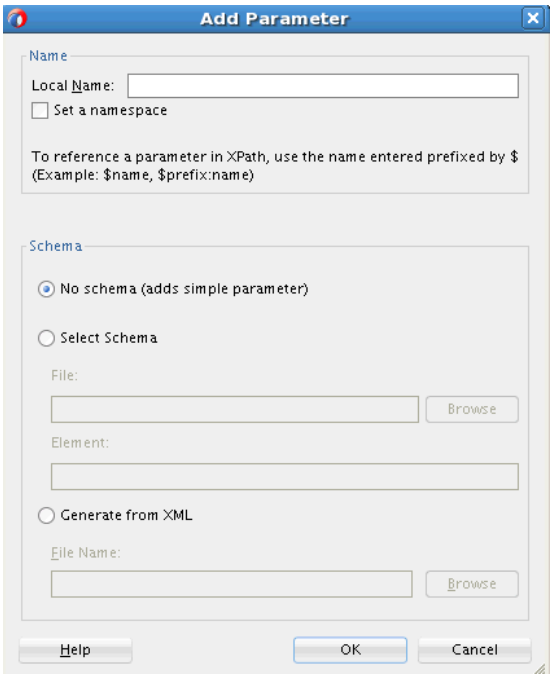
Figure 40-43 Adding Global Parameter from the XSLT Toolbar



You can also right-click anywhere on the canvas (center) pane, and select **Create > Add Global Parameter** from the context menu.

The Add Parameter dialog appears. Figure 40-44 shows the Add Parameter dialog.

Figure 40-44 Add Parameter Dialog



2. Enter a Local Name for the parameter and optionally specify a namespace.

   If the parameter is a complex parameter, you can specify a schema and an element definition for the parameter. Click the **Help** button in the dialog to get more information on the individual fields.

3. Click **OK** in the Add Parameter dialog to create the parameter.

The parameter node appears at the appropriate place in the target pane (for Map View) or XSLT pane (for the XSLT View). This enables you to map XPath expressions to the parameter to set the parameter's default value.

The parameter also appears in the source tree. This enables you to map the parameter to XPath expressions or nodes in the XSLT tree.

Note:

You can also add parameters like other XSLT elements. See How to Add XSLT Statements for details about adding XSLT elements using the context menu or Components window.

Substituting Elements and Types

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.

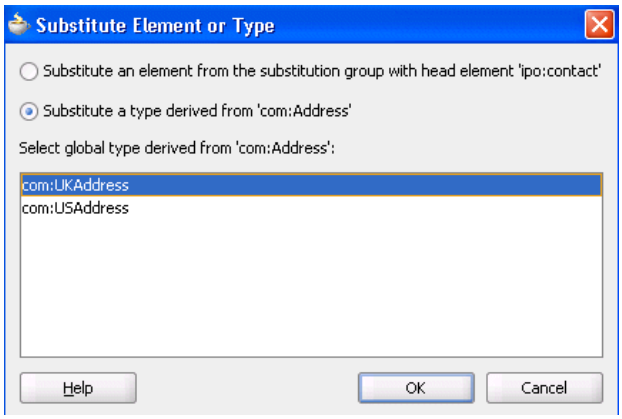
Type substitution is supported by use of the `xsi:type` attribute in XML.

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. If you are working in the XSLT pane, the element you select must exist in the XSLT before substitution.
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 40-45 appears.

Figure 40-45 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 the type of substitution and where the substitution is done, as described in the following sections:

   - For Type Substitutions

     – Type substitutions in the source tree

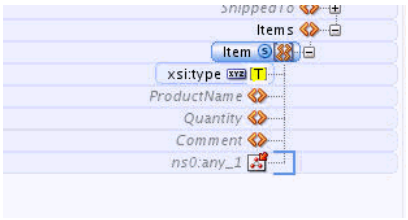
     The **xsi:type** attribute is added beneath the original element, as shown in the preceding figure. An **S** icon is displayed against the element to indicate that the node was substituted. You can map from any structural elements in the substituted type, including the **xsi:type** attribute.

Note:

Unlike element substitution, only one type substitution at a time can be displayed in the source tree. However, this does not prevent you from writing a map that allows the source to switch between the original type and the substituted type.

If a node is not visible in the source tree, and the node is mapped to an XPath expression, the XPath expression mapped to the node is still displayed in the center canvas pane.

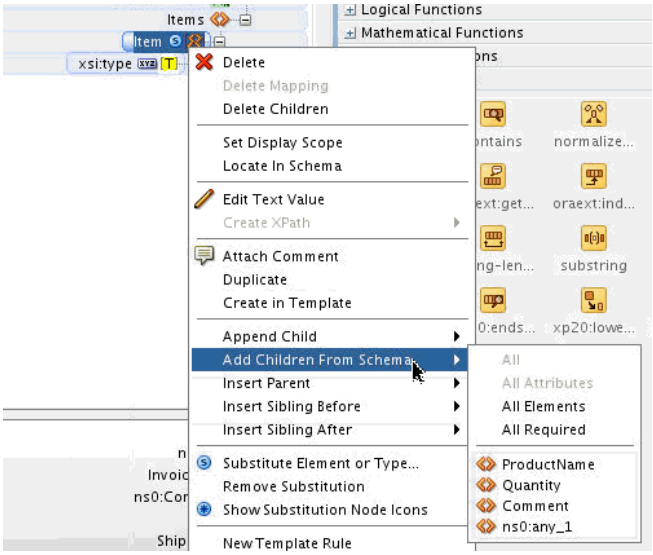
     – Type substitutions in the Map View target tree



     The **xsi:type** attribute is added beneath the original element, as shown in the preceding figure. The attribute is disabled in Map View, and set to the type value that was selected. An **S** icon is displayed against the element to indicate that the node was substituted. You can map to any structural elements in the substituted type, except the **xsi:type** attribute.

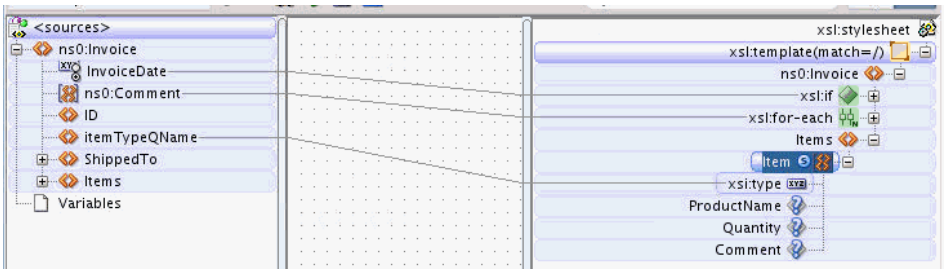
     – Type substitutions in the XSLT pane of the XSLT View

The **xsi:type** attribute is added beneath the original element. It's value is set to the type value that was selected, but may be mapped to. An **S** icon is displayed against the element to indicate that the node was substituted. You can add any structural elements through the **Add Children From Schema** context menu option.



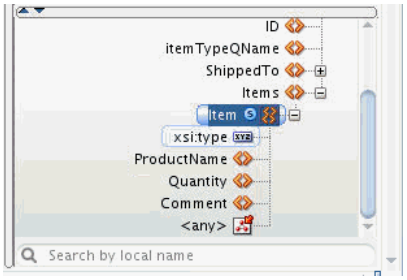
In some cases, it may be necessary to set the value of the **xsi:type** field dynamically using an XPath statement. If you need to dynamically set the value of the **xsi:type,** you can use type substitution to temporarily provide access to the structural elements that are needed for the expected value at runtime.

Add the elements that are needed, then map the desired XPath statement to the **xsi:type** attribute to set the value dynamically. As the runtime value for **xsi:type** is not available at design time, question-mark icons (**?**)are displayed on elements that depend upon the type value, if it is set dynamically.



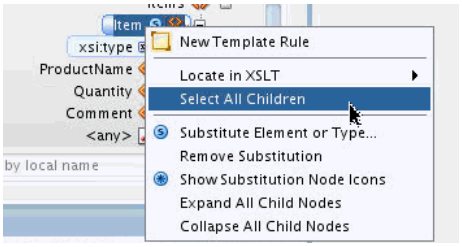
– Type substitutions in the target pane of the XSLT View

  You can also make substitutions in the target pane of the XSLT View. This pane represents the target schema document. After you make a type substitution in the target pane, the **xsi:type** attribute is added beneath the original element along with any structural elements associated with that type, as shown in the following figure.

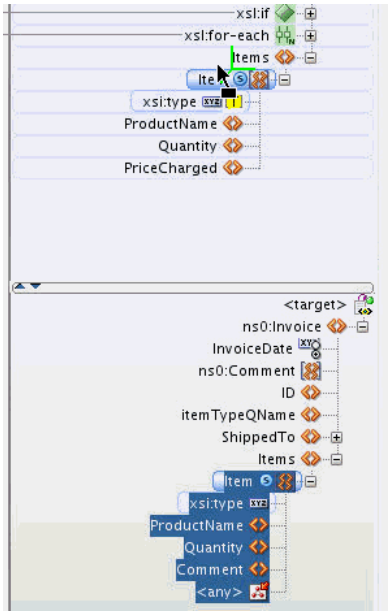


You can select these elements and drop them into the XSLT pane, as needed. These elements also show up in the **Add Children From Schema** context menu option available in the XSLT pane.

For example, in the following figure, we select all children of a substituted element (**Item**).



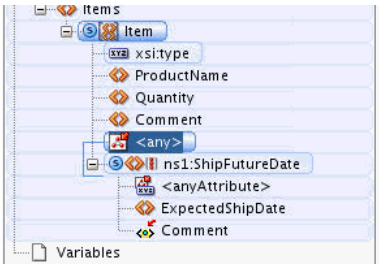
We then drop the new **Item** node as a child of the **Items** node in the XSLT pane, as shown in the following figure.



The Item node and its children are added as children of the Items node. You can similarly create different structures from different substitutions in the target pane.

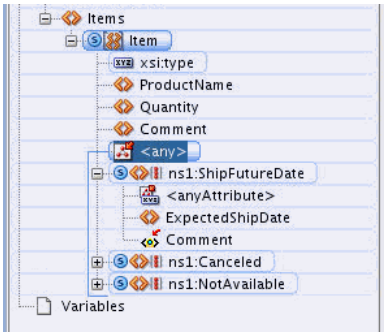
- For Element Substitutions
  - Element substitutions in the source tree

Both the original element and the substituted element are displayed in the source tree, and are connected by a blue bracket. An **S** icon is displayed against the node that is substituted. You can map from any structural elements in the substituted element.

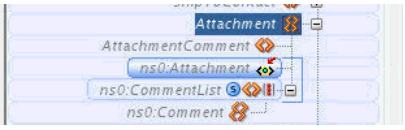


In the preceding figure, the **ns1:ShipFutureDate** is substituted for the **any** element.

You can also substitute multiple elements at the same time, as shown in the following figure.



– Element substitutions in the Map View target tree

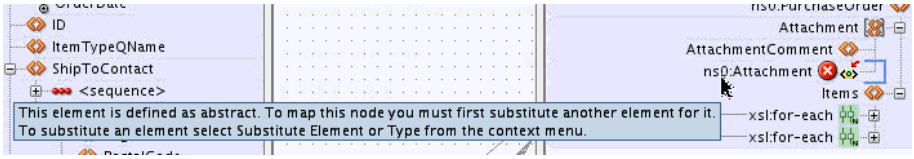


As shown in the preceding figure, both the original element and the substituted element are connected with a blue bracket. An **S** icon is displayed against the node that was substituted. You may map to any structural elements in the substituted element.

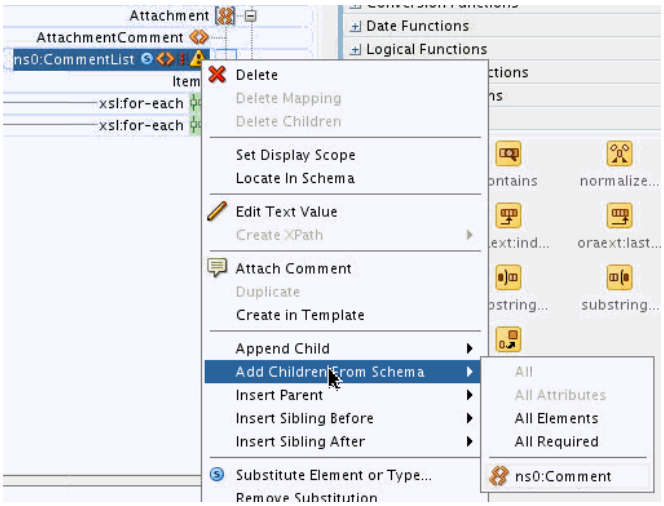
– Element substitutions in the XSLT pane of the XSLT View

In order to substitute an element in the XSLT pane, the original element must be one that can appear in the XSLT. Any elements cannot appear in the XSLT pane, and must be substituted in the XSLT View target pane, as discussed in the next section. Abstract elements can be added to the XSLT pane temporarily, but should not be used as final output. Elements that are the head of a substitution group and are not abstract can be used as normal elements, and also be substituted.

In the following figure, the ns0:attachment element is an abstract element that is also the head of a substitution group. When a substitution is made for this in the XSLT pane, the element is replaced with the substitution.

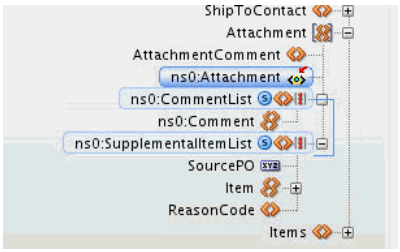


After substitution, the abstract element is replaced with the selected element. The **S** icon indicates the substitution. You can add child elements to the substituted element using the **Add Children From Schema** context menu. This is depicted in the following figure.



– Element substitutions in the target pane of the XSLT View

You can also make substitutions in the target pane of the XSLT View. The target pane represents the target schema document. After making an element substitution in the target pane, the elements substituted are added beneath the original element along with any structural elements associated with that type, as shown in the following figure.



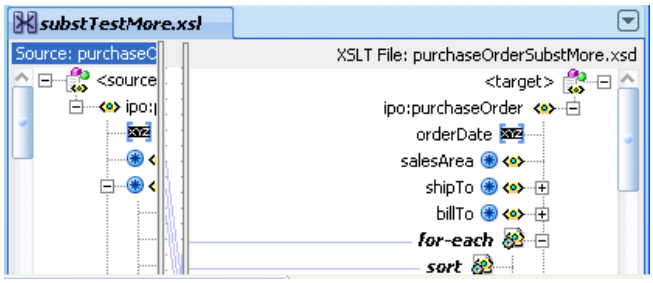
You can select these elements and drop them into the XSLT pane, as needed. These elements also show up in the **Add Children From Schema** context menu option available in the XSLT pane.

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 40-46.

Figure 40-46 All Possible Substitutions



8. To hide the icons, right-click and select **Hide Substitution Node Icons**.

Using Named Templates

You can add named templates to the XSLT map. These templates can be edited within the XSLT Map Editor. You can invoke named templates by using the `xsl:call-template` instruction.

Named templates can only be used with the XSLT View. Once you add a named template in an XSLT map, the map can only be opened in XSLT View.

How to Create a Named Template

To create a new named template:

1. Right-click a blank area in the XSLT pane, and select **New Named Template** from the context menu. Alternatively:

   - Right-click a blank area in the canvas (center) pane, and select **Create >New Named Template** from the context menu.
   - Right-click any source node in the source pane, and select **New Named Template** from the context menu.
   - Select the Add button, identified by the green plus (+) icon, in the XSLT toolbar, and select **New Named Template.**

   The Add Named Template dialog appears.

2. Enter a name for the template. Optionally, set a namespace.

   You can click **Help** to display help on the available options in the dialog.

3. Optionally click the **Add** button, identified by the green plus (+) icon, to add a parameter. The **Add Parameter** dialog appears.

4. Enter a name for the parameter. Optionally, set a namespace.

   If the parameter is a complex parameter, you can specify a schema and an element definition for the parameter. Click the **Help** button in the dialog to get more information on the individual fields.

5. Click **OK** in the Add Parameter dialog to add the parameter.

6. Add any more parameters required for the named template.

7. Click **OK** in the Add Named Template dialog to create the named template.

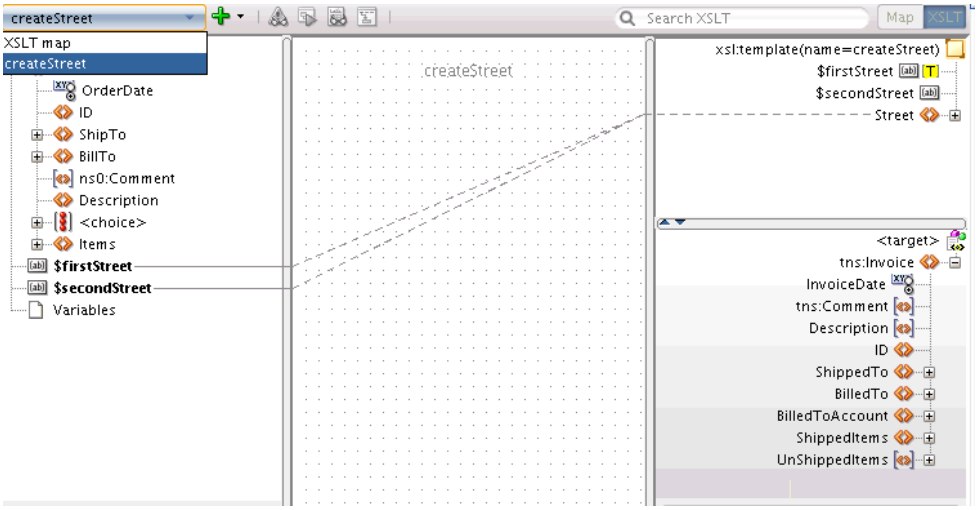
How to Edit a Named Template

When a named template is first created, it is opened for editing in the XSLT Map Editor. You can switch between editing the named template and editing the XSLT map by using the drop-down list in the XSLT toolbar.

Figure 40-47 shows a named template being edited in the XSLT Map Editor. The top-left hand corner has a drop-down list that lets you choose between the XSLT map and the named template (createStreet).

The source tree, any global parameters, and the named template parameters appear in the source pane on the left. The XSLT pane on the upper right represents the XSLT associated with the named template. If a target schema is used, then the target tree appears at the bottom-right corner of the editor.

Figure 40-47 Editing a Named Template



How to Add Parameters to an Existing Named Template

To add parameters to an existing named template:

1. Ensure that the named template appears in the XSLT Map Editor. To switch to the named template, select the name of the named template from the drop-down list in the XSLT toolbar. See Figure 40-47 for an example.
2. Right-click a source node and select **Add Parameter** from the context menu. Alternatively:
   - Right-click in a blank area on the canvas (center) pane, and select **Create > Add Parameter** from the context menu.
   - Click the Add button, identified by the green plus (+) icon, on the XSLT toolbar. Select **Add Parameter** from the drop-down list that appears.
3. Specify a name for the parameter, and other details, in the Add Parameter dialog. Click the **Help** button in the dialog to get more information on the individual fields.
4. Click **OK** in the Add Parameter dialog to add the parameter.

How to Invoke a Named Template

A named template is invoked using the `xsl:call-template` instruction. You can add the `xsl:call-template` instruction as a node in the XSLT pane.

To invoke a named template:

1. Add the `xsl:call-template` instruction as an XSLT node. You can add the `xsl:call-template` instruction from the context menu or the Component window. See How to Add XSLT Statements for details about adding XSLT elements.

   The **Set Attribute** dialog appears.

2. Select the named template to be invoked. Click **OK.**

   The `xsl:call-template` instruction is added to the XSLT tree.The parameters (`xsl:use-param` instructions) are added as child nodes.

3. Map values to the `xsl:with-param` XSLT nodes to set the values for the parameters.

Using Template Rules

Template rules are `xsl:template` statements with match attributes. Template rules are supported by the XSLT Map Editor. You can use template rules in the XSLT View only. Template rules are not supported in the Map View.

The XSLT Map Editor enables you to add template rules in various ways. You can insert the template rule manually, or refactor an existing mapping to create a template rule.

When adding the template rule manually, you also need to insert an `apply-templates` statement to invoke the template at the appropriate place in the XSLT. When refactoring an existing map to create a template rule, the `apply-templates` statement is inserted automatically.

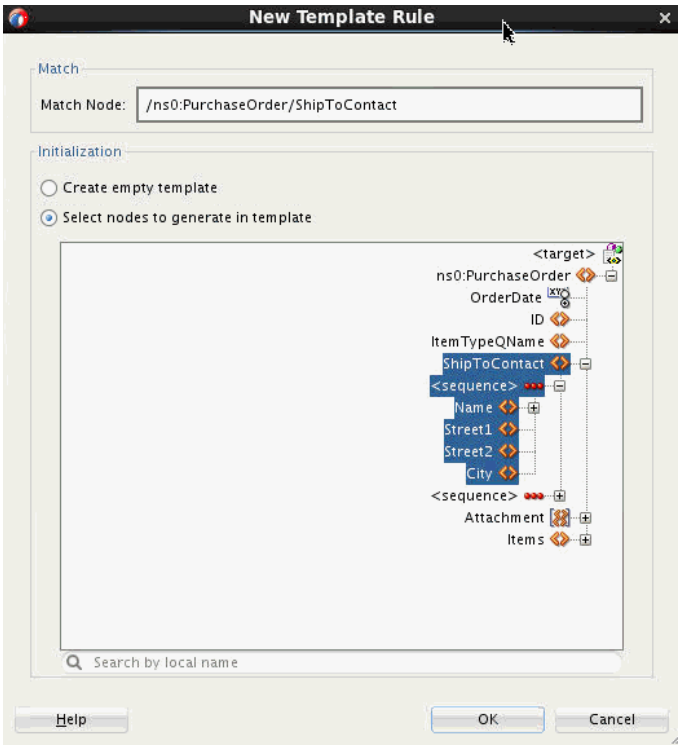
How to Create a Template Rule

1. Use one of the following methods to invoke the New Template Rule dialog:

   - Right-click the node in the source pane that is to serve as the match node. Select **New Template Rule** from the context menu that appears.
   - Right-click a node in the target pane. Select **New Template Rule** from the context menu that appears.
   - Right-click a node in the XSLT pane. Select **New Template Rule** from the context menu that appears.
   - Right-click a blank area in the canvas (center) pane. Select **Create > New Template Rule** from the context menu that appears.
   - Click the **Add** icon, identified by the green plus (+) sign, on the XSLT toolbar. Select **New Template Rule** from the list of options that appears.

   The New Template Rule dialog appears. Figure 40-48 shows the New Template Rule dialog.

Figure 40-48 New Template Rule Dialog



The New Template Rule dialog contains the following fields:

- **Match Node:** Specifies the value for the match attribute in the `xsl:template` definition.

  The match attribute contains a pattern used to match a node in the input XML document. The XSLT processor executes the instructions within a template when the node it is processing matches the pattern defined in the template match attribute.

- **Initialization Section:** Used to determine the content of the new template rule.

  You can choose to select **Create empty template** to create an `xsl:template` instruction with no content. Alternatively, you can select **Select nodes to generate in template** to view and select target schema nodes that you would like to create when the template is executed.

  The **Select nodes to generate in template** option is available only if a target schema is being used.

- **Search by local name:** If the **Select nodes to generate in template** option is selected, a tree representing the target schema is displayed. This option enables you to search for a node in the target schema tree using its local name.

The New Template Rule dialog box may have automatically populated fields, depending on the mode you choose to invoke the dialog. Table 40-2 lists the pre-populated fields corresponding to each choice. If the method of invocation is not listed, then no fields are pre-populated.

Table 40-2 Methods of Invoking the New Template Rule Dialog

Invoked From	Automatically Populated Information
A node in the source pane	The **Match Node** is populated with the absolute path to the source node. You can edit this value, if required. The **Create Emty Template** option is pre-selected in the Initialization section. You may change this selection, if required.
A node in the target pane	The **Select nodes to generate in template** option is pre-selected in the Initialization section and the target schema tree is displayed. The target schema node used to invoke the New Template Rule dialog is pre-selected in the target tree. You may change this selection, if desired. The **Match Node** field must be populated with the desired match pattern for the template.

2. Set the **Match Node** as desired. Here are some common examples:

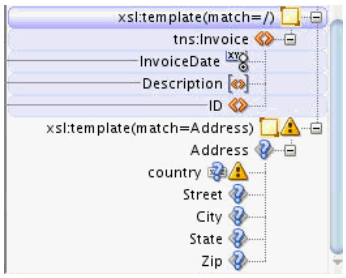
   - `/ns0:PurchaseOrder/ns0:ShipToContact/ns0:Region`: The template executes when the processor is processing the node with this path.
   - `Item`: The template executes when the processor is processing any node with the name `Item.`
   - `HighPriorityItems/Items/Item`: The template executes when the processor is processing any `Item` node that is a child of an `Items` node that in turn is a child of a `HighPriorityItems` node.

3. Select the content desired in the Initialization section. If you know the output nodes that you would like the template to create, select the **Select nodes to generate in template** option, and select the output nodes from the target schema tree that is displayed. You can use the **Search by local name** field to search within the target schema tree. You can select multiple nodes in the target tree by clicking each desired node while holding down the Shift key.

4. Click **OK** in the New Template Rule dialog to create the template. A new `xsl:template` statement is inserted at the end of the XSLT.

5. If you do not have pre-existing `apply-templates` statement that invokes the template rule, then insert an appropriate `apply-templates` statement to invoke the template rule.

   If an `apply-templates` statement is not present, you can see a warning icon against the `xsl:template` statement and question-mark (**?**) icons against any nodes within the template. The following figure shows an example:

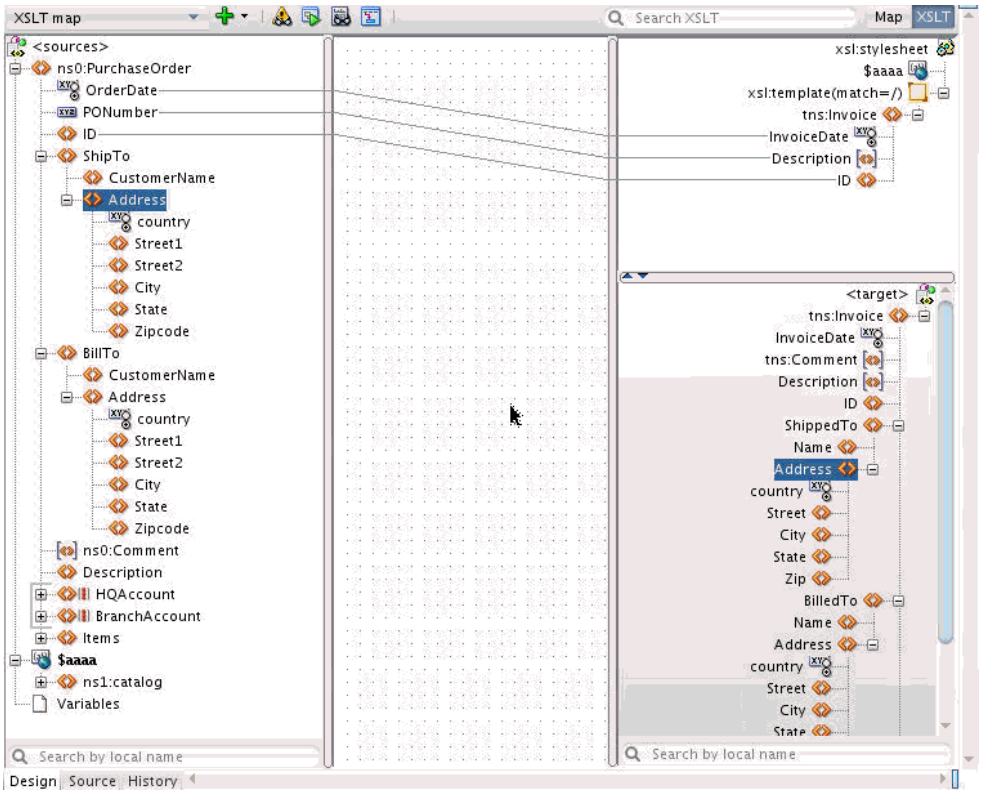


In general, the `apply-templates` statement is inserted at the position where you would like to generate the nodes contained in the template. For example, in the preceding figure, the `apply-templates` statement needs to be inserted at the point where the Address node and its children need to be created.

After you insert the `apply-templates` statement to invoke the template rule, the warning icon and the question mark (**?**) icons disappear.

Example: Creating a Template Rule

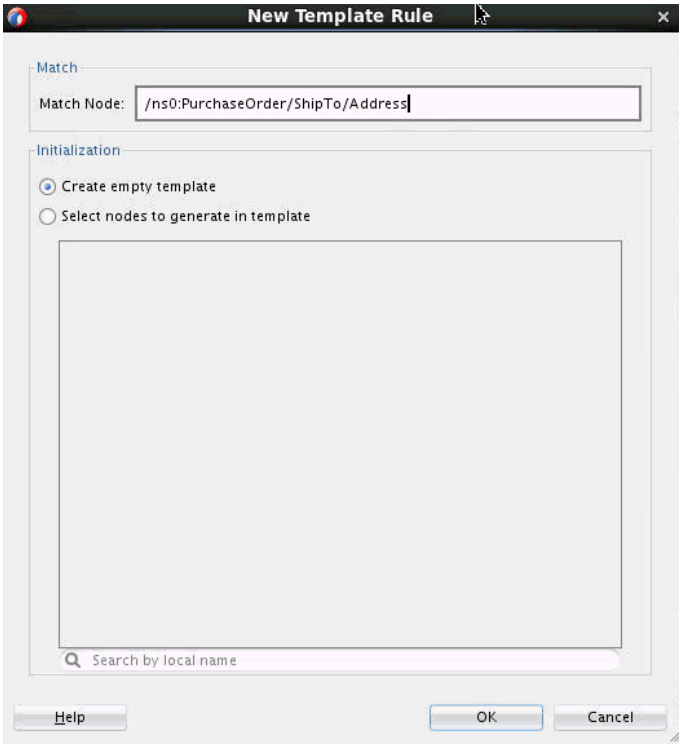
As an example, let us create a template rule for an existing map, as shown below:



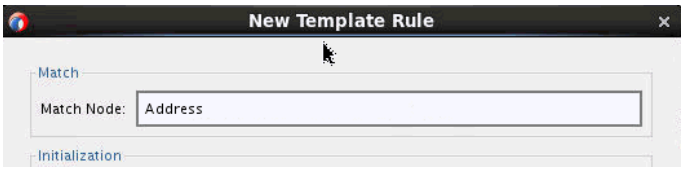
This example seeks to create a single template that processes the **Address** information in the source **ShipTo** and **BillTo** elements to create the **Address** in the **ShippedTo** and **BilledTo** elements in the target. The following steps illustrate the process:

1. Right-click the **Address** node under the **ShipTo** or **BillTo** node in the source pane, and select **New Template Rule** from the context menu that appears.

   The New Template Rule dialog appears. The **Match Node** is pre-populated with the path to the **Address** node that you selected in the source pane. This match string is specific to the **Address** element under **ShipTo,** but we need to create a template that will process any **Address** field in the source document.



2. Modify the **Match Node** string to contain only the name **Address.** This will match all **Address** nodes in the source document irrespective of where they appear in the document.

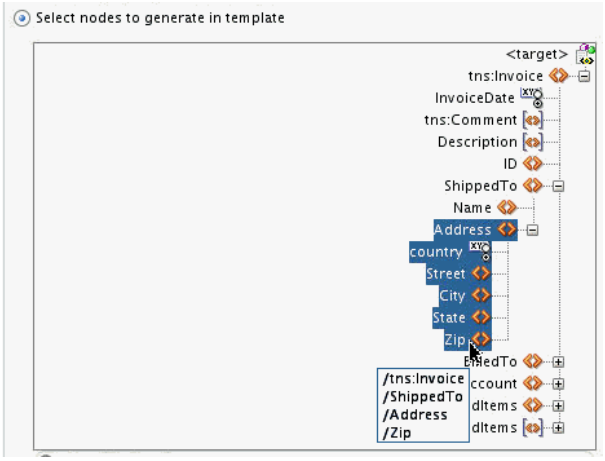


3. Select the **Select nodes to generate in template** option. The target schema tree appears.

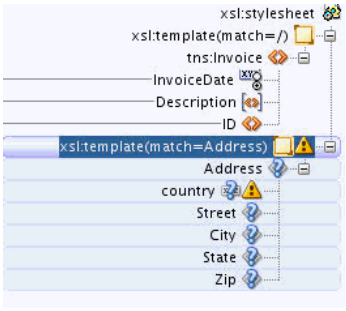
   When the template rule is invoked, we would like to create the **Address** element in the target, along with its children.

4. Select the **Address** node under the **ShippedTo** or the **BilledTo** element, as both elements have identical structures.

5. To select the children of the Address node, press and hold down the Shift key, and click the Zip element, which is the last child of the Address element. This selects the Address element and all its children.



6. Click **OK** to create the template rule.

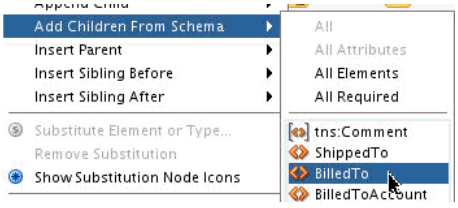


As the rule is not yet invoked anywhere in the map, a warning icon appears against the template, and the nodes that the template creates have question mark (**?**) icons against them.

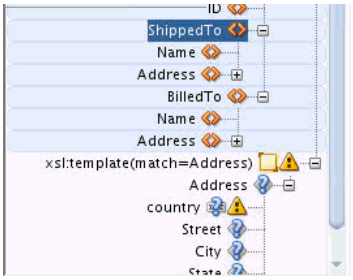
Invoking the Template

Next, we need to invoke the template for both the **ShipTo** and **BillTo Address** elements in the source to create the **ShippedTo** and **BilledTo Address** elements in the target. We need to create `apply-templates` statements in the XSLT at the places where we would like to create these **Address** elements. The following steps describe the process.

1. Right-click the **Invoice** node in the XSLT pane, and select **Add Children From Schema > BilledTo** from the context menu that appears. The **BilledTo** node is inserted along with its required child nodes.

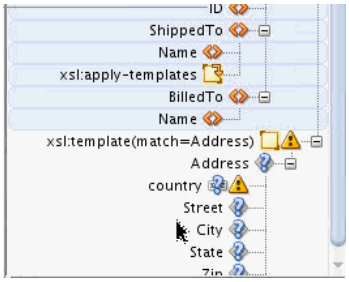


2. Right-click the **Invoice** node in the XSLT pane, and select **Add Children From Schema > ShippedTo** from the context menu that appears. The **ShippedTo** node is inserted along with its required child nodes.



3. Right-click the **ShippedTo/Address** node and select **Delete.** Repeat the same for the **BilledTo/Address** node. We would create the **Address** nodes using the template rule that we created.

4. Right-click the **ShippedTo** node and select **Append Child > XSL > apply-templates** from the context menu that appears. **The xsl:apply-templates** statement is added.

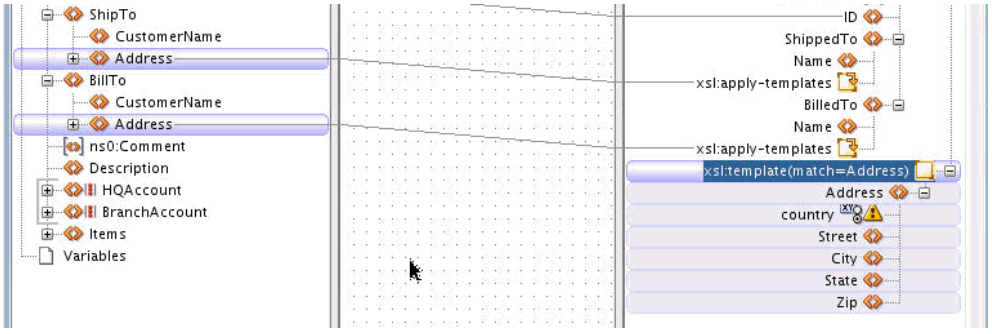


5. Right-click the **BilledTo** node and select **Append Child > XSL > apply-templates** from the context menu that appears. **The xsl:apply-templates** statement is added.

6. Drag a line from the S**hipTo/Address** node in the source pane to the **ShippedTo/ apply-templates** node in the XSLT pane. This sets the select attribute of the `apply-templates` statement, so that only the **ShipTo/Address** node is processed by the `xsl:apply-templates` statement.

7. Drag a line from the **BillTo/Address** node in the source pane to the **BilledTo/ apply-templates** node in the XSLT pane. This sets the select attribute of the `apply-templates` statement, so that only the **BillTo/Address** node is processed by the `xsl:apply-templates` statement.

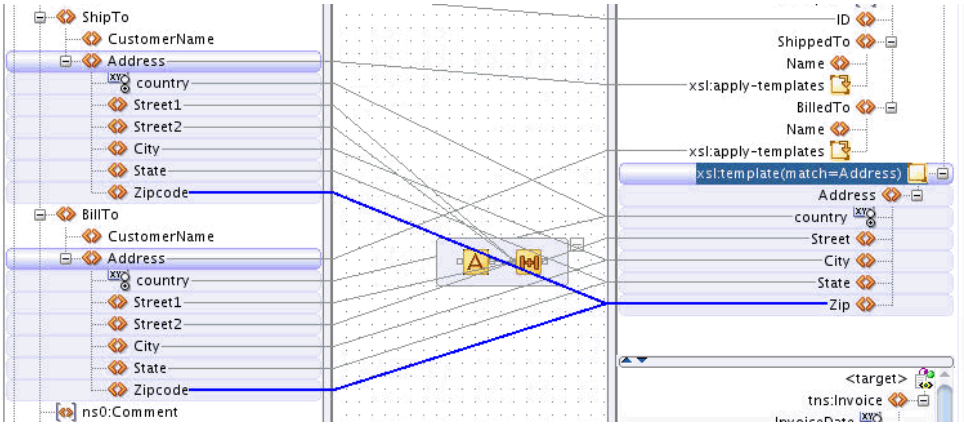
   At this point, the warning icon on the template rule disappears, as we have defined the template invocation. If you click the template rule, the two source **Address** nodes processed by the template are highlighted, as illustrated in the following figure.



Next, the nodes below the template rule can be mapped.

8. Drag and drop lines from the elements under the **BillTo/Address** node, or the **ShipTo/Address** node, to the appropriate elements under the **Address** template rule.

   As you drag from either source **Address** (**BillTo** or **ShipTo**), lines are drawn to both source addresses. This is because both **BillTo/Address** and **ShipTo/Address** are context nodes for the template.



The source code for the template now appears as follows:

```
<xsl:template match="Address">
  <Address country="{@country}">
    <Street>
      <xsl:value-of select="concat (Street1, ', ' , Street2 )"/>
    </Street>
    <City>
      <xsl:value-of select="City"/>
    </City>
    <State>
      <xsl:value-of select="State"/>
    </State>
    <Zip>
      <xsl:value-of select="Zipcode"/>
    </Zip>
  </Address>
</xsl:template>
```

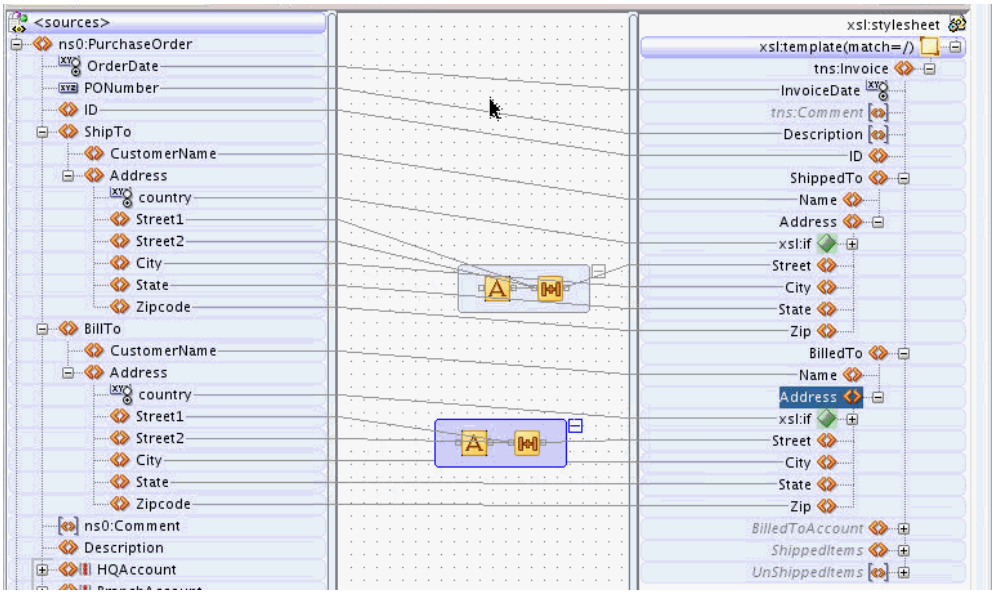
How to Refactor an Existing Map to Create a Template Rule

You can refactor code from an existing template to create a new template rule. This is useful if the template rule that is created can be reused in multiple places, as was the case with the Address template rule created in the previous section.

The **Create in Template** option enables you to refactor a section of XSLT instructions into a separate template that can be invoked from multiple places.

In the following example, we refactor an existing mapping to create a template rule.

In the following map, the XSLT that processes each Address element in the source is repeated for each Address node. For easier maintenance, you may want to consolidate redundant code into reusable templates. This way, if the code is later updated, you would not have to update multiple copies.

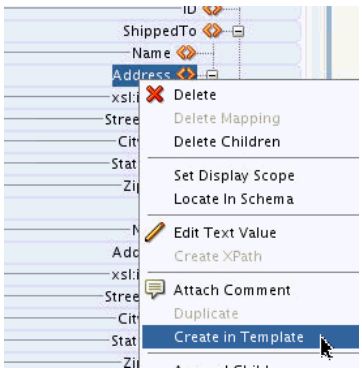


In the preceding map, if the XPath `concat` expression that creates **Street** from **Street1** and **Street2** needs to be modified, there are two copies that would have to be modified. However, if the `concat` function exists in a single template that is reused to produce both **Address** elements, then only one `concat` statement needs to be modified in future.

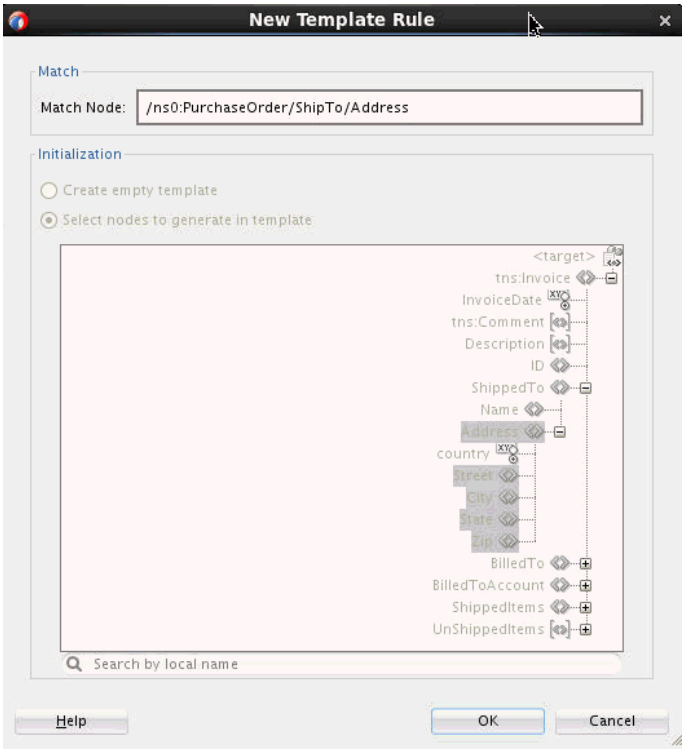
In the following steps, we refactor the existing mapping to create a single template that processes the **Address** elements:

1. Make sure that you are in the XSLT View. You can click **XSLT** in the top right corner of the XSLT Map Editor to switch to the XSLT View.

2. Right-click the **ShippedTo/Address** element in the XSLT pane and select **Create in Template** from the context menu that appears.



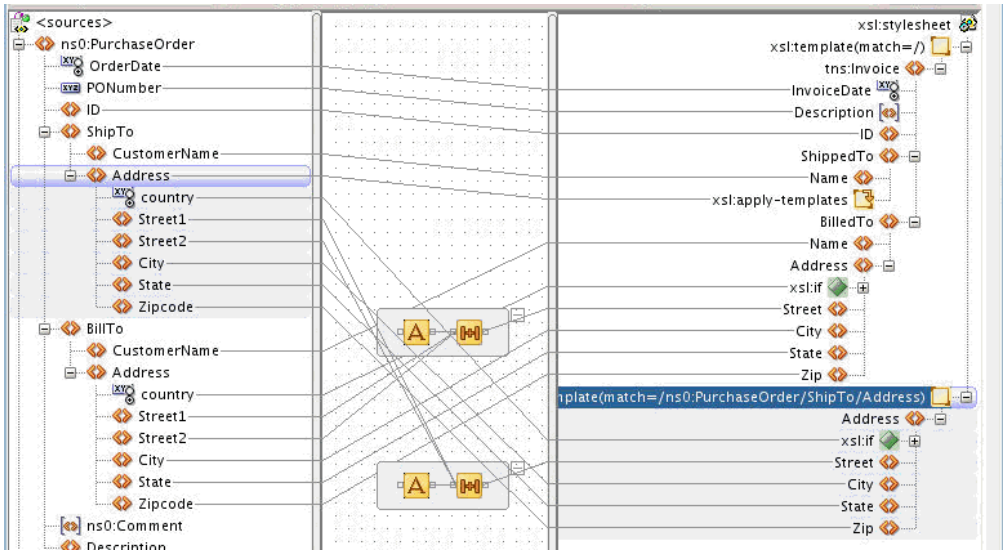
The New Template Rule dialog appears.



The **Match Node** is pre-populated with a suggested match pattern derived from XPath expressions contained under the selected node in the XSLT pane. The elements selected in the target schema tree are the Address node and its children. These elements would be moved to a new template rule.

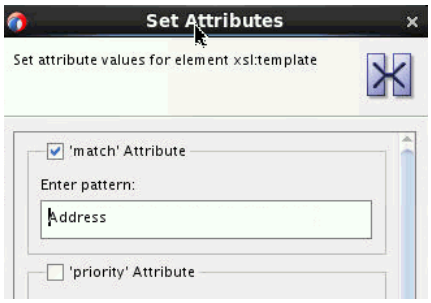
3. Click **OK.**

A new template rule is created and an `apply-templates` statement is inserted in place of the **Address** node. The XPath expressions defined for the elements under the Address node in the XSLT pane are updated to contain relative paths to the Address node (context node) for the new template.

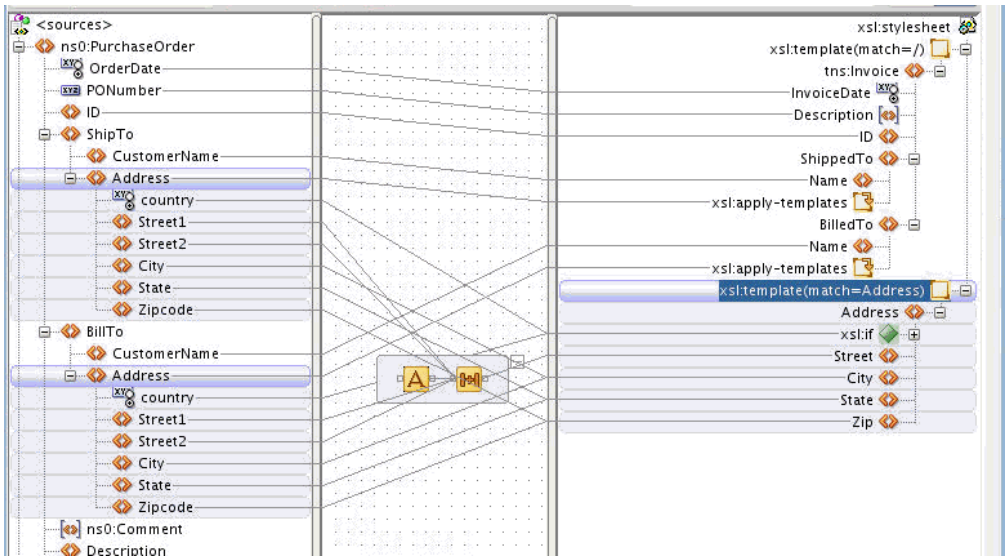


Next, we use this template for both the **ShipTo** and **BillTo Address** elements.

4. Double-click the new template node. The Set Attributes dialog appears.
5. Under **Enter Pattern,** enter Address.



6. Click **OK.**
7. Delete the **BilledTo/Address** element in the XSLT pane.
8. Right-click the **BilledTo** node and select **Add Child > XSL > apply-templates**. This creates an **xsl:apply-templates** statement in place of the **Address** node.
9. Drag a line from the **BillTo/Address** node in the source pane to the new **BilledTo/apply-templates** node in the XSLT pane.



Both **BilledTo** and **ShippedTo Address** elements are now created using a single template rule.

Using the Execution View

The Execution View displays the order of execution of your XSLT statements. When creating complex XSLT that uses named templates and template rules, it can get difficult to determine the order of execution of XSLT templates. The Execution View helps you troubleshoot issues by creating an execution tree for your XSLT.

The execution tree shows when the output nodes are created, and shows exactly which templates are invoked at various points during the XSLT execution.

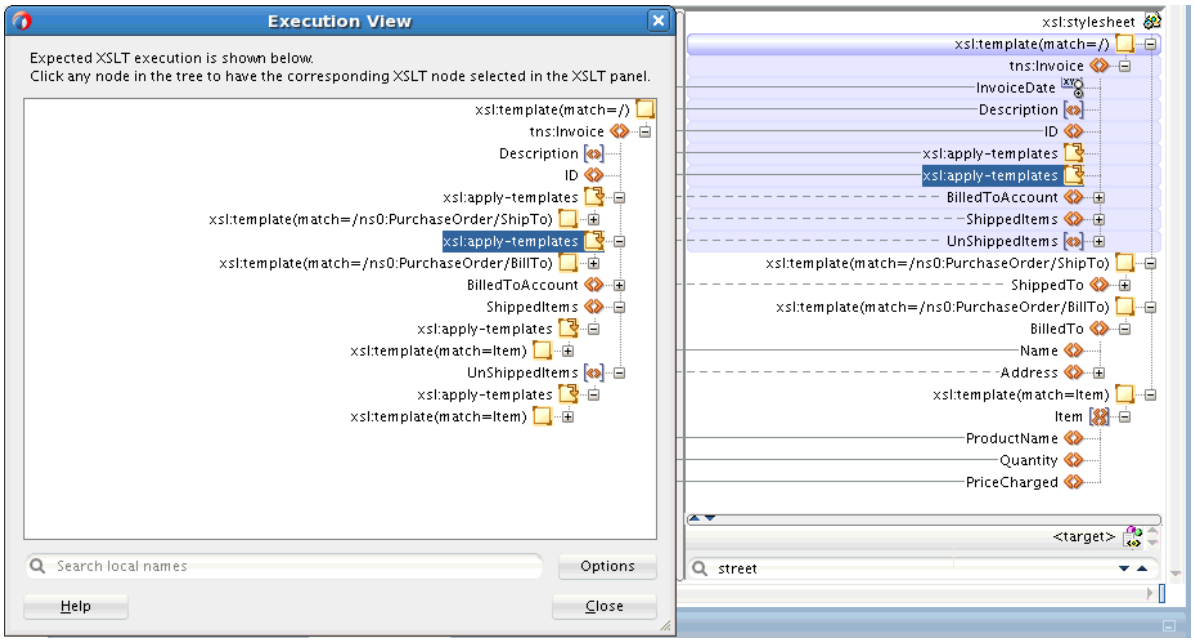
How to Use Execution View to Prevent or Troubleshoot Runtime Errors

To launch the Execution View dialog, right-click the canvas (center) pane, and select **Execution View.** Alternatively, click the **Execution View for XSLT Map** button on the XSLT toolbar.

The Execution View is available in both the Map and XSLT views. However, it is most useful when the map contains multiple templates and the user needs to figure out as to where the templates are being invoked.

Figure 40-49 shows the Execution View dialog for an XSLT map that contains multiple templates. As per the execution view, the root template with `match='/'` executes first. This is followed by the creation of the `Invoice, Description,` and `ID` nodes. An `apply-templates` statement then invokes the **ShipTo** template followed by an `apply-templates` statement that invokes the **BillTo** template. Other nodes and templates are then created.

Figure 40-49 Execution View Dialog and Corresponding XSLT Tree



If you click an element in the Execution View tree, the corresponding element is highlighted in the XSLT tree. For example, as shown in Figure 40-49, if you want to locate the `apply-templates` node that invokes the **BillTo** template, select the `apply-templates` node in the Execution View and the corresponding `apply-templates` statement is highlighted in the XSLT pane.

Execution View also shows calls to named templates. When you select a node inside a named template call in the Execution View tree, the XSLT editor view is refreshed to show the selected node in the named template implementation.

Execution View helps you understand the overall flow of an XSLT stylesheet. Using the Execution View, you can locate issues related to templates that are not invoked, or `apply-templates` statements that are invoking incorrect templates.

When using imported named templates or template rules, Execution View shows the corresponding calls, and you can use Execution View to troubleshoot problems like import precedence. However, Execution View cannot navigate to the external XSLT files.

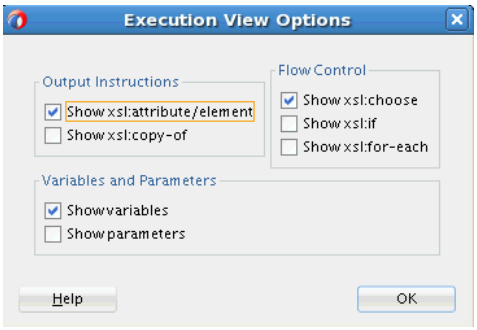
Searching for Nodes

The Execution View dialog has a search facility that enables you to search for specific nodes in the Execution View tree. Click in the **Search local names** field, and type a name to search.

Setting Display Options

You can choose to show or hide certain XSLT elements in the execution tree. Click Options to bring up the Execution View Options dialog. Figure 40-50 shows the default selections in Execution View Options.

Figure 40-50 Default Execution View Options



Troubleshooting Memory Issues

If you work with large schema documents, you may sometimes encounter an out-of-memory error during auto-mapping, or during test and report generation. If you receive an out-of-memory error when using the XSLT Map Editor, you must increase the heap size of the JVM to resolve the problem.

To increase the JVM heap size:

1. Locate the config file for your application installation.

   Locate the shared Oracle JDeveloper product.conf file or the optional tool-specific .conf file located in the user's home directory. The location of these files depends on the host platform.

   - For Windows Platforms:

     The location of user/product files is often configured during installation, but may be found here:

     `%APPDATA%\JDeveloper\`*`product-version`*`\product.conf`

     `%APPDATA%\JDeveloper\`*`product-version`*`\jdev.conf`

   - For UNIX Platforms:

     `$HOME/.jdeveloper/`*`product-version`*`/product.conf`

     `$HOME/.jdeveloper/`*`product-version`*`/jdev.conf`

2. Edit the file to change the `AddVMOption` to the desired value. For example:

```
AddVMOption -Xmx1024M
```

Note:

The `AddVMOption` value can be large for 64-bit machines. Setting it close to the amount of RAM provided by the machine provides better performance.

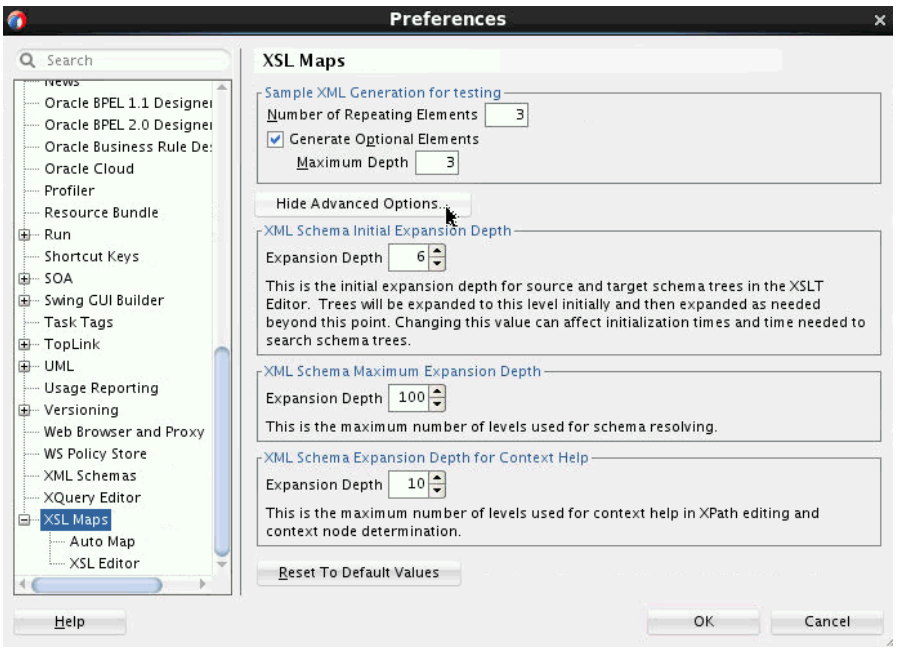
Setting XSL Map Preferences

Use the Preferences dialog to set preferences related to XSLT maps, such as expansion depth, and the XSLT Map Editor, such as initialization options.

How to Set XSLT Map Preferences

To set XSLT map preferences, select **Preferences** from the **Tools** menu. Click **XSL Maps** in the navigation tree that appears in the left pane of the Preferences dialog. Figure 40-51 shows the XSL Maps dialog that appears.

Figure 40-51 XSL Maps Preferences

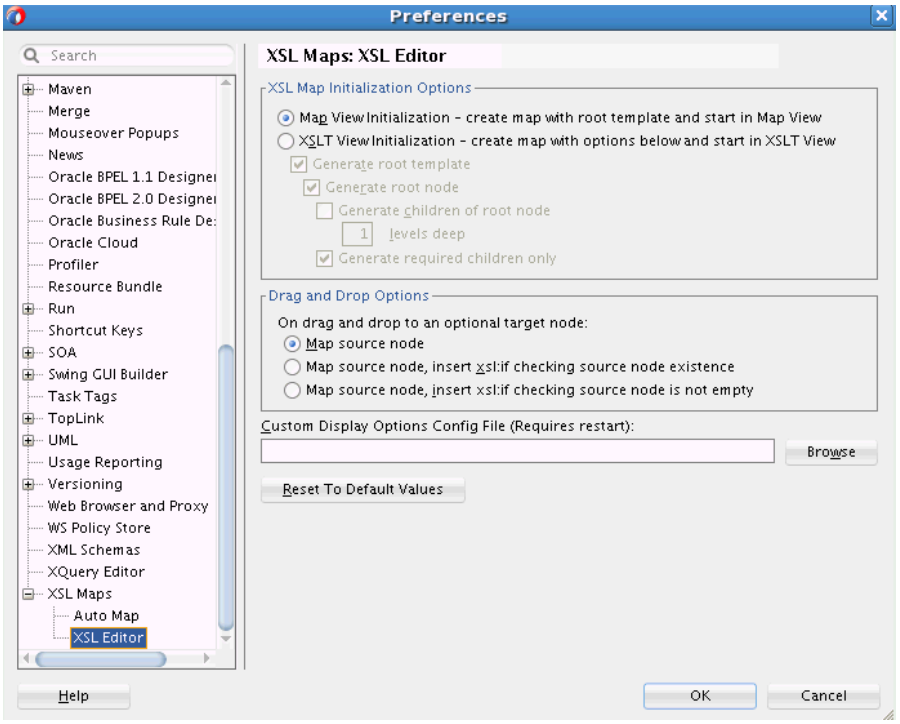


You can set various options such as the expansion depth of the source and target trees. You can click the **Help** button for more information on each field.

How to Set the XSL Editor Preferences

To set XSLT map preferences, select **Preferences** from the **Tools** menu. In the navigation tree that appears in the left pane of the Preferences dialog, click the plus sign (+) next to **XSL Maps.** This expands the XSL Maps node. Select XSL Editor. Figure 40-52 shows the XSL Editor Preferences dialog.

Figure 40-52 XSL Editor Preferences



You can set various preferences for the XSLT Map Editors, such as the initialization mode. Click **Help** for more information on each field.

How to Import a Customization File to Specify Display Preferences in the XSLT Map Editor

You can specify a customization file containing display preferences for the XSLT Map Editor. In the XSL Editor Preferences dialog (Figure 40-52), click the **Browse** button to the right of the Custom Display Options Config File field. Select the file to import.

The customization file is an XML file that must conform to the `XSLTEditorOptions.xsd` schema located in the `bpm-ide-common.jar` file at:

`oracle/tip/tools/ide/common/resource/XSLTEditorOptions.xsd`

The following example shows a sample customization file. The important elements in the file are described following the example.

```
<?xml version="1.0" encoding="UTF-8" ?>
  <customizeXSLTeditor>
      <selectedArea>HL7</selectedArea>
      <abbreviationLists>
          <abbreviationList name="Siebel">
              <abbreviation long="Account" short="Acct"/>
          </abbreviationList>
          <abbreviationList name="SalesGeneral">
              <abbreviation long="SalesOrder" short="SO"/>
              <abbreviation long="PurchaseOrder" short="PO"/>
              <abbreviation long="BillOfMaterial" short="BOM"/>
              <abbreviation long="CreateEngineeringChangeOrderList" short="CECOL"/>
          </abbreviationList>
      </abbreviationLists>
      <customization area="HL7">
          <showFixedValueInElementName>
```

```
            <path>@LongName</path>
            <path>@LongName2</path>
            <path>@Name</path>
        </showFixedValueInElementName>
        <abbreviations>
           <apply display="treeLabels">
               <hideText part="matchValue" maxLength="13" hide="left"/>
               <hideText part="namedTemplateName" maxLength="15" hide="left"/>
               <hideText part="importHref" maxLength="20" hide="left"/>
           </apply>
           <apply display="dropDownLists">
               <hideText part="namedTemplateName" maxLength="40" hide="center"/>
           </apply>
        </abbreviations>
    </customization>
    <customization area="AIA">
        <abbreviations>
           <applyAbbreviations list="Siebel"/>
           <applyAbbreviations list="SalesGeneral"/>
           <apply display="treeLabels">
               <hideText part="matchValue" maxLength="13" hide="left"/>
               <hideText part="namedTemplateName" maxLength="15" hide="left"/>
               <hideText part="importHref" maxLength="20" hide="left"/>
           </apply>
           <apply display="dropDownLists">
               <hideText part="namedTemplateName" maxLength="40" hide="center"/>
           </apply>
        </abbreviations>
    </customization>
</customizeXSLTeditor>
```

The following list describes the important elements in the preceding example:

- `<selectedArea>:` Selects the customization area to be used by the editor. A list of customization areas may be defined in the file.
- `<abbreviationList name="listName">:` Defines an abbreviation list that can be referenced by a customization area.
- `<abbreviation name="Account" short="Acct">:` Defines a specific abbreviation to use in an abbreviation list.
- `<abbreviations>:` Used within a customization area to define abbreviations and cut-off lengths for text in the editor.
- `<applyAbbreviations>:` Selects an abbreviation list or lists to use in this customization area.
- `<apply display="treeLabels" | "dropDownLists">:` Selects an area where text cut-offs occur.
- `<hideText>:` Selects specific text fields to cut-off when they are too long.
- `@part:` Either `"matchValue", "namedTemplateName",` or `"importHref".`
- `@maxLength:` Text value length limit.
- `@hide:` Specifies portion of the text to hide, `"left"` truncates the text on the left side, `"right"` truncates the text on the right side, `"center"` removes text in the center replacing it with '...'.

- `<showFixedValueInElementName>:` Used within a customization area. This element selects fixed value attributes that contain the long name or other text that the user wants to see displayed as part of the element name in the editor source or target tree.

  For example:

  ```
  <showFixedValueInElementName>
     <path>@LongName</path>
     <path>@LongName2</path>
     <path>@Name</path>
  </showFixedValueInElementName>
  ```

  In the preceding example, the first fixed attribute found on any element in the XSLT Map Editor trees with the name `LongName, LongName2,` or `Name` is shown as part of the element tree name.

  The fixed attribute value is shown in parentheses to the right of the actual element name in the tree. This is particularly useful for HL7 schemas where descriptive names are added as fixed attribute values in the schema.

41

Creating Transformations with the XQuery Mapper

This chapter describes how to create, edit, and test XQuery transformations using the XQuery Mapper. The XQuery Mapper enables you to transform data between various XML and non-XML types, enabling you to integrate heterogeneous applications rapidly. You can use the XQuery (.xqy) files created using XQuery Mapper as resources in Oracle BPEL Process Manager, Oracle Mediator, or Oracle Service Bus.

This chapter includes the following sections:

Introduction to the XQuery Mapper

The XQuery Mapper supports XQuery 1.0. The older XQuery 2004 is also supported.

The XQuery Mapper includes the following views:

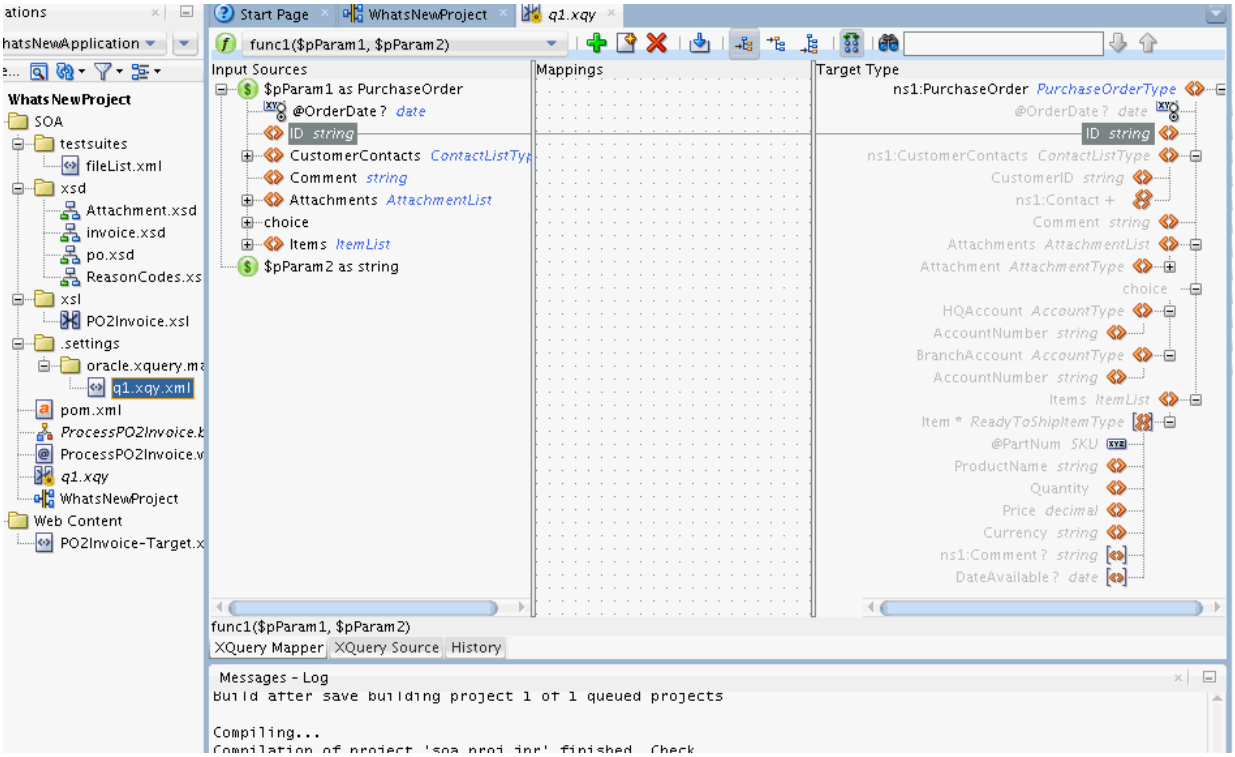
- XQuery Mapper Graphical View
- XQuery Mapper Source Editor

> **Note:**
>
> The XQuery Mapper graphical view is not supported for XQuery 2004 files. Only the source view is supported for this older XQuery version.

When you create a new XQuery file, it opens in the graphical view by default. The graphical view can also be accessed by clicking the **XQuery Mapper** tab at the bottom of the XQuery map.

Figure 41-1 shows the graphical view of the XQuery mapper.

Figure 41-1 XQuery Mapper



The left pane of the XQuery Mapper includes the input sources or parameters for the XQuery function. If your XQuery file has multiple functions, you can choose the function to display using the toolbar over the mapper panes.

The right pane includes the target schema tree, which corresponds to the XQuery function's result type. The center pane helps you map the source and the target schema elements using XQuery functions.

About the Source and Target Trees

The left pane of the XQuery Mapper shows the source tree, and the right pane shows the target tree. Tree nodes can be XML elements, attributes, and some other XQuery constructs.

XML elements are identified by the <> icon. Attributes use a different icon, and attribute names are prefixed with the @ symbol, as they appear in an XPath expression. The element or attribute multiplicity is shown using the following standard suffixes:

- **?**: Zero or one occurrence of an element/attribute.
- **+**: One or more occurrences of an element/attribute.
- ***:** Zero or more occurrences of an element/attribute.

The source tree shows the input sources or parameters for the selected XQuery function. The root level elements represent the input parameters for the function. If a root node is a complex element, then its child elements and attributes appear under the root node.

The target tree can include XML elements, attributes, and some programming control structures. The elements and attributes can appear in the following forms:

- Grayed Font: An element that is part of the target schema, but not defined yet. Once you map a grayed element to a source element, it appears in normal font.
- Normal Font: An element that either corresponds to an element constructor in the source, or copied implicitly from the source data.
- Underlined Font: An element that is incompatible with the specified target schema. This element may appear because of an incorrect element name used in an element constructor, or because of a sequence assignment with an incorrect schema type.

The target tree can also contain programming control structures like If-Then-Else, Union operator, and comma operator:

- If-Then-Else Operator: The If-Then-Else operator shows up as a node called *Conditional*. The Conditional node has nested branches for If Then and Else. You can choose the **Make Conditional** option from the context menu of a node to make it conditional.
- Union Operator (and other sequence combining operators): These cannot be created in the graphical view of the XQuery Mapper. However, if the source view contains such an operator, it is represented in the target tree with a node called *All*, and the operands are represented as subnodes of the all node.
- Comma Operator: The comma operator shows up as a node called List. The subnodes represent the comma-limited operands of the comma operator. You can choose the **Clone** option from the context menu of target tree node to apply a comma operator.

Using the XQuery Mapper Toolbar

The XQuery Mapper toolbar is located above the XQuery Mapper panes. The toolbar contains various tools to work with the graphical mapper. Figure shows the XQuery Mapper toolbar.

Figure 41-2 XQuery Mapper Toolbar

The XQuery Mapper toolbar contains the following tools:

- **Function Selector:** The function selector box is identified by a green icon with the letter *f* on it. You can use the Function Selector to select the function to display in the source pane. This is useful if your XQuery map contains multiple functions.
- **Add New Function:** The Add New Function button is identified by the green plus (+) sign. Use Add New Function to add a new function to the XQuery map file.
- **Rename Function:** The Rename Function button is to the right of the Add New Function button. Use the Rename Function button to rename a function in the XQuery map file.
- **Delete Function:** The Delete Function is identified by a red cross (X) sign. Use Delete Function to delete a function from the XQuery map file.
- **Import Library Module:** The Import Library Module button is to the right of the Delete Function button. Use Import Library Module to import a library XQuery function into the map. You must specify the library module files to be imported.

- **Mapping Mode:** The XQuery mapper can use different mapping modes. These modes affect the XQuery expressions created when the user drags and drops a line from a source node to a target node. The next three buttons are used to select the corresponding mapping mode:
  - **Value Mapping:** Constructs target XML elements and attributes from the input source, and copies the input source values, using XML constructors. For example:

    ```
    <ID>{fn:data($pParam1/ID)}</ID>
    ```

    The above code creates the `ID` element in the target from the `ID` element in the input source parameter.

    Value mapping is the default mapping mode.
  - **Overwrite Mapping:** The XML elements from the input source parameters are copied to the result sequence. Any existing mapping are replaced with the new mapping. For example:

    ```
    {
    $pParam1/Items
    }
    ```

    The above code copies the Items subtree, together with its child elements and attributes, to the target tree.
  - **Append Mapping:** This mode works like overwrite mapping, except that any existing mappings are not overwritten. New additional mapping are created.

  See Using the XQuery Mapper for more information on using the mapping modes.
- **Show/Hide Target Type Differences:** Use the Show/Hide Target Type Differences button to manage the visibility of XML elements and attributes in the right target tree. You can choose to hide elements and attributes that haven't been mapped yet.
- **Search:** Use the Search field to search for elements, attributes, data types, and so on in the source and target trees. Use the Up and down arrows to look for the next and previous items respectively.

Using the Properties Window

The Properties window displays the XQuery expression for the node selected in the target tree. XQuery expressions created using drag and drop can be edited in the Properties window. The Properties window can also be used to create more complex XQuery expressions.

The Properties window is located below the XQuery Mapper, by default. If the Properties window is not visible, click **Properties** under the JDeveloper **Window** menu to display the Properties window.

Tip:

When working with the XQuery Mapper, you might want to move the Properties window from the bottom right hand corner of the screen to the bottom of the screen, directly below the mapper window. A larger Properties window makes it easy to edit XQuery expressions and view the variable tree.

You can directly edit the XQuery expression, for the selected target node, in the Properties window. You can also drag XQuery functions, constructs, and operators from the Components window into your XQuery expression in the Properties window. The Components window is located to the right of the XQuery Mapper, by default. If you cannot see the Components window, select **Window > Components** from the Oracle JDeveloper menu bar.

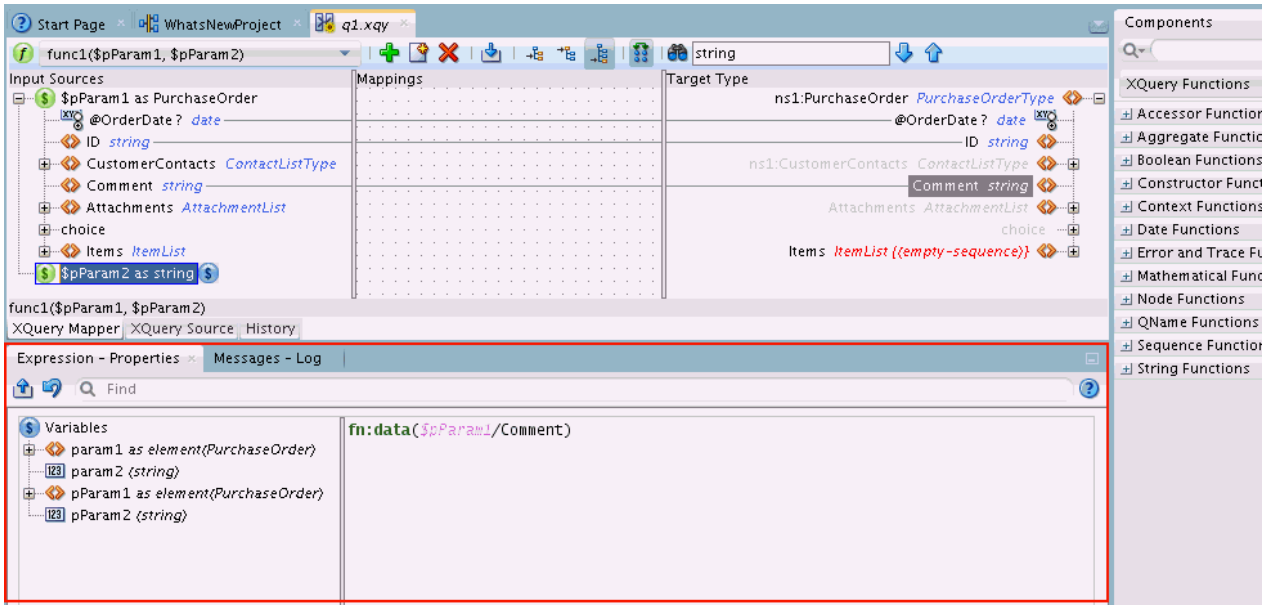
To save the changes, click the **Commit** button in the top left corner of the Properties window. The XQuery is recompiled, and the XQuery Mapper view is updated.

If you make an error when editing the XQuery expression, click **Revert to mapper sources**, in the top left area of the Properties window, to undo the changes and start again.

The Properties window also includes a variable tree on the left hand side. The variable tree shows all variables, both local variables and XQuery function parameters, that are visible in the current scope. The scope is determined by the node highlighted in the target tree pane. You can drag and drop nodes from the variable tree into your XQuery expression in the right pane.

Figure 41-3 shows the Properties window. The Properties window is highlighted in red. The comment node is shown selected in the target tree. The corresponding variable tree and XQuery expression (`fn:data($pParam1/Comment)`) appears in the Properties window.

Figure 41-3 Properties Window



Using the Components Window

The Components window contains all the XQuery functions and operators that you can use in your XQuery maps. These functions and operators can be dragged and dropped to the center pane of the XQuery Mapper. You can also drag and drop a function onto a target tree node, if the target tree node has already been created using the **Insert** context menu option.

Note:

You can also drag functions and operators to an XQuery expression in the Properties window, as described in the preceding section.

When a function is dragged and dropped on an existing link between a source and target node, for example, it becomes a part of the expression corresponding to that link. Some functions without parameters must be dragged to an empty area of the center pane, and associated with a target node. Functions can also be chained together.

The Components window organizes the XQuery function and operators into the following categories:

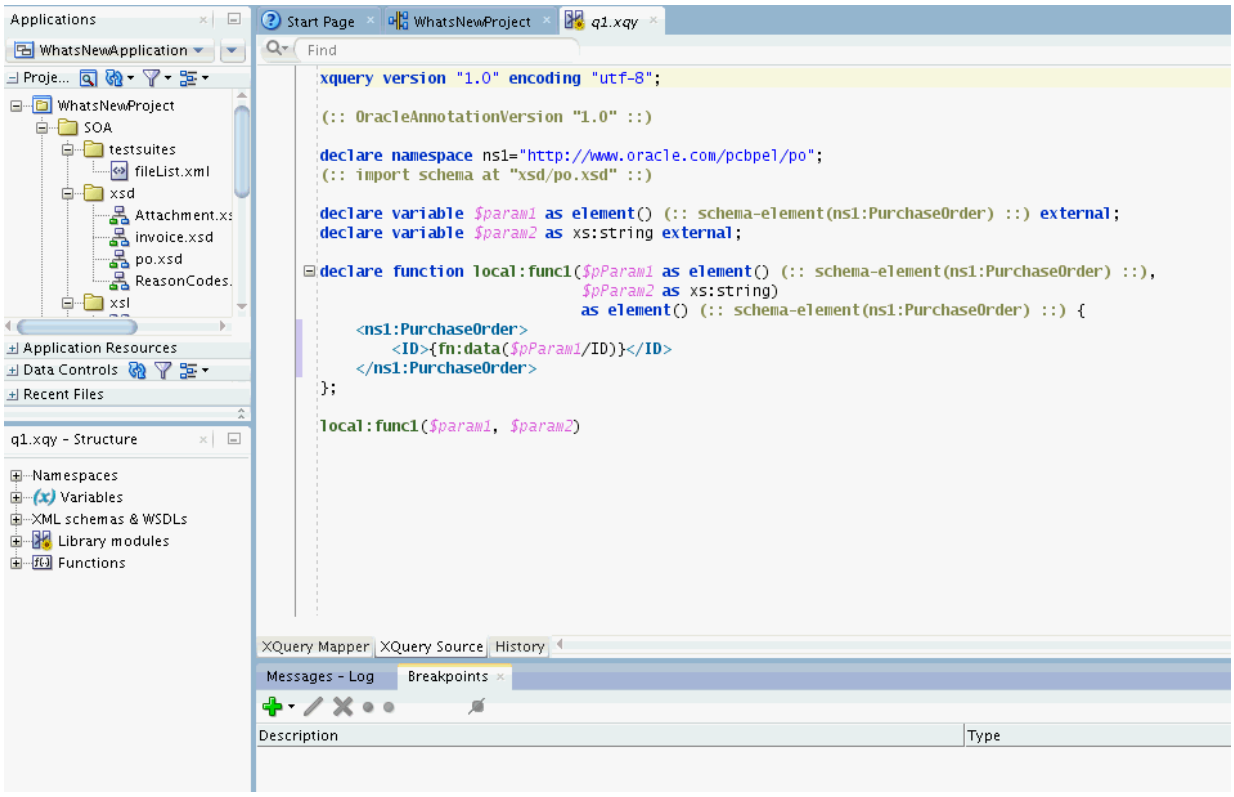
- **XQuery Functions:** Includes various categories of XQuery functions, like aggregate functions, date functions, mathematical functions, string functions, and so on.
- **XQuery Constructs:** Includes standard XQuery constructs like If-Then-Else and FLWOR constructs.
- **XQuery Operators:** Includes various categories of XQuery operators, like logical operators, node comparison operators, and so on.
- **User-Defined Functions:** Includes all the functions that you have defined in the current XQuery map file, and any functions from imported library modules.
- **My Components:** Includes your favorite components that you can add to this category. It also includes the recently used functions.

Source Editor

The source editor enables you to edit the XQuery map directly, and also allows you to perform tasks that cannot be directly performed in the graphical view.

Click the XQuery Source tab at the bottom left of the XQuery Mapper graphical view to display the source editor. Figure 41-4 shows the XQuery Mapper source editor.

Figure 41-4 XQuery Mapper Source Editor



The XQuery source view provides code editing features like code highlighting, code completion, error highlighting, and code folding. You can also use Ctrl + click (click the left mouse button while holding down the Ctrl key) on a function name, variable name, schema, or schema element to navigate to the corresponding declaration for the function, variable, schema, or schema element respectively.

Creating an XQuery Map File

Use Oracle JDeveloper to create XQuery maps. XQuery maps are included in the project as .xqy files.

XQuery maps can be created as main modules and library modules. A main module is an executable XQuery file. A library module is used to group and store XQuery functions. When you import a library module into a main module, all functions in the library module become available in the main module.

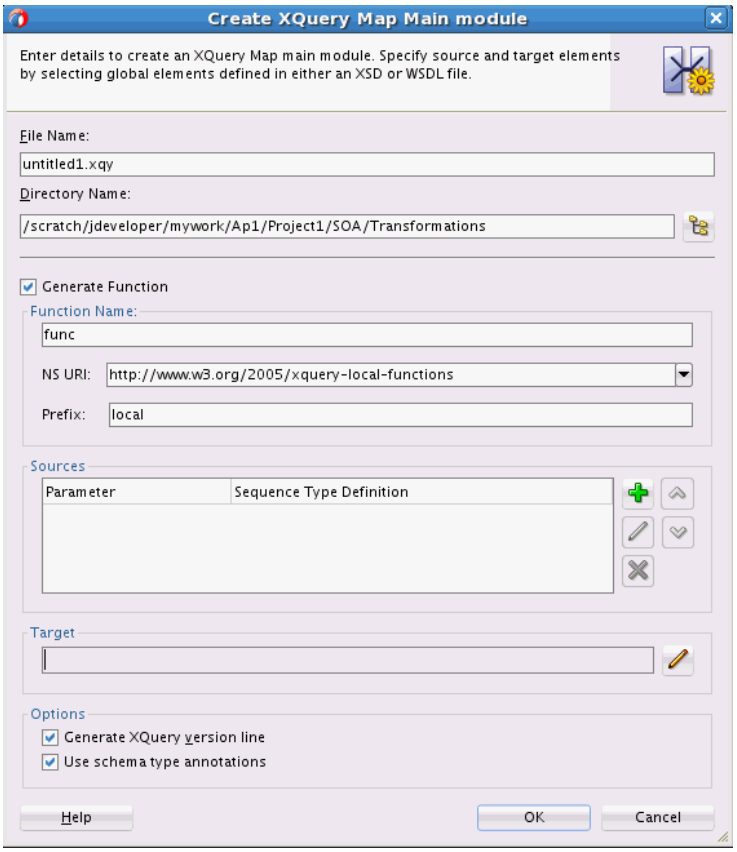
How to Create an XQuery Main/Library Module

To create an XQuery Main/Library Module:

1. Click the **File** menu. Select one of the following:

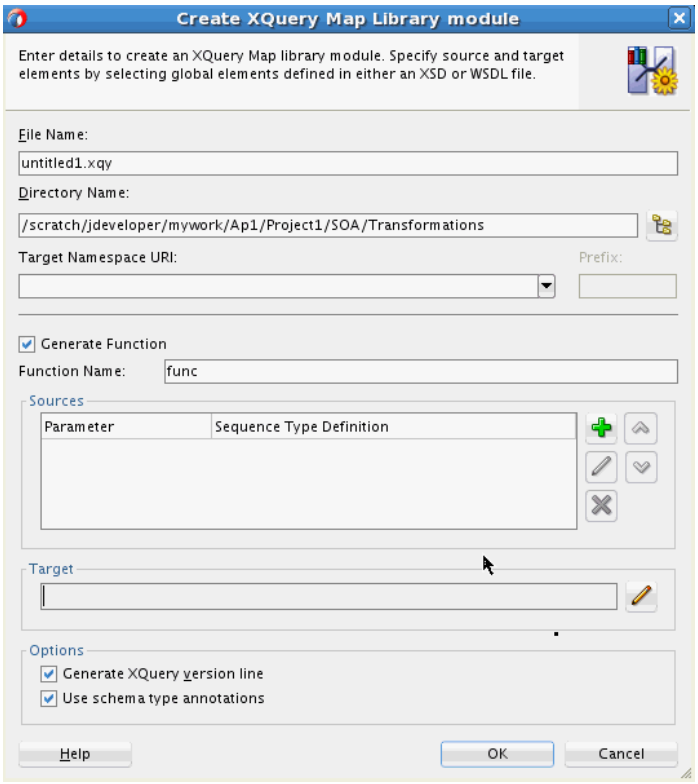
   - To create an XQuery main module, select **New > XQuery File ver 1.0.** The **Create XQuery Map Main Module** dialog box appears.

Figure 41-5 Create Main Module Dialog



- To create an XQuery library module, select **New > XQuery Library ver 1.0.** The **Create XQuery Map Library Module** dialog box appears.

Figure 41-6 Create Library Module Dialog



2. Under **File Name,** enter the name for the XQuery map file to be created. The file must have a `.xqy` extension.

3. Under **Directory Name,** specify the directory in which the map file should be created. This is usually the Transformations directory in your project folder. You can click the tree icon on the right to browse and select the directory of your choice.

4. If you are creating a library module, select the target namespace for the library module under **Target Namespace URI.** Optionally edit the **Prefix** for the namespace.

   Every function defined in a library module automatically uses the library's target namespace.

5. Select **Generate Function** to create a function in the XQuery file. If you do not select this, an empty XQuery file is created, and you can add functions later.

6. Under **Function Name,** enter the name of the function to be created in the XQuery file.

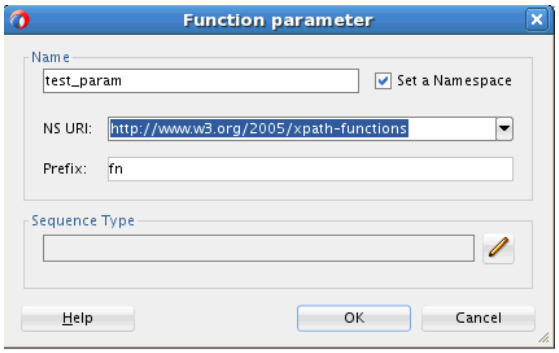
7. If you are creating a main module, select these additional fields for the function:

   - **NS URI** specifies the namespace for the function. **NS URI** is automatically populated. You can also select a different namespace.

   - **Prefix** specifies the namespace prefix of the function. **Prefix** is populated automatically. You can also edit the suggested namespace prefix.

8. Add parameters for the function under the **Sources** section. To add a parameter, click the Add Source button identified by the **green plus sign** (+). The **Function Parameter** dialog box appears.

Figure 41-7 Function Parameter Dialog



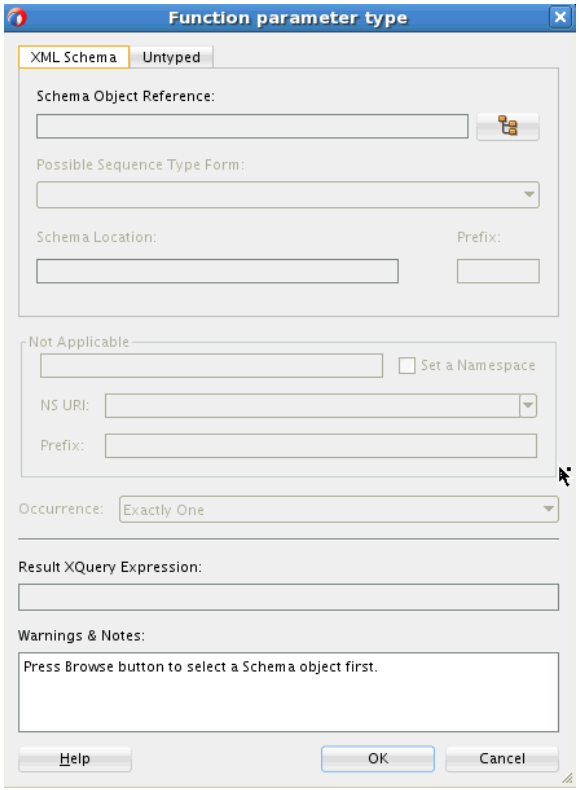
9. Under **Name**, enter the name of the function parameter.

10. Select **Set a Namespace** to specify a namespace for the function parameter:

    Under **NS URI**, select the namespace for the function parameter. The namespace prefix appears in the **Prefix** field. You can optionally edit this.

11. Under **Sequence Type**, click the button identified by the **pencil** icon to specify the data type for the parameter. The **Function Parameter Type** dialog box appears.

Figure 41-8 Function Parameter Type Dialog



Use the **XML Schema** tab to specify an XML schema type as the data type for the function parameter. The **Untyped** tab can be used to specify an untyped (non-XML

schema based) form of the parameter. You use an XML schema type in this procedure.

12. In the Function Parameter Type dialog, click the button to the right of **Schema Object Reference** (identified by the tree icon) to select a schema object as the data type.

    This brings up the **Type Chooser** dialog box. You can choose from **Project Schema Files, XML Schema Simple Types,** and schemas embedded in **Project WSDL Files**. Navigate to the desired XML type and click **OK** to close the Type Chooser dialog box.

13. In the Function Parameter Type dialog, the **Possible Sequence Type Form, Schema Location,** and **Prefix** are automatically populated depending on your choice of **Schema Object Reference.** Optionally change any values if required.

14. Under **Occurrence**, optionally change the multiplicity of the parameter. The resultant XQuery expression appears under **Result XQuery Expression.**

15. Click **OK** to close the Function Parameter Type dialog box.

16. Click **OK** to close the Function Parameter dialog box.

17. In the Create XQuery Map Main Module/Library Module dialog, specify the function's result data type under the **Target** section. Click the button, with the **pencil** icon, to the right of the **Target** field.

    The Function Result Type dialog box appears. This dialog box is identical to the Function Parameter Type dialog box. Use instructions under Steps 12 to 14 to specify the function's result data type.

18. Click OK to close the **Function Result Type** dialog box.

19. In the Create XQuery Map Main Module/Library Module dialog box, under the **Options** section, select **Generate XQuery version line** to generate a standard line at the beginning of the XQuery file.

    For example, the following line might be generated at the beginning of the file:

    ```
    xquery version "1.0" encoding "utf-8";
    ```

20. Select **Use schema type annotations** to create a weak-typed XQuery file that uses type annotations in place of schema object references. This may improve the XQuery performance for certain scenarios.

    If you deselect this option, XQuery generates a strong-typed XQuery file that can contain references to schema objects.

    See Using Type Annotations to Improve XQuery Performance for more information on type annotations.

21. Click **OK**. The newly created XQuery map opens up in the **XQuery Mapper** graphical view. If you want to see the XQuery source editor, click **XQuery Source.**

Using the XQuery Mapper

This section contains the following topics:

How to Use Value Mapping to Copy a Leaf Element Value to a Target Leaf Element

To create a value map for a leaf element:

1. Make sure that Value Mapping mode is selected in the XQuery toolbar.
2. Select the source leaf element whose value needs to be copied.
3. Hold down the left mouse button, and drag the mouse pointer to the target leaf element. Release the left mouse button.

   A solid line connecting the source and target leaf node appears. The source leaf element is now value-mapped to the target leaf element.

How to Use Overwrite Mapping to Copy an Element Subtree to the Target Tree

To create an overwrite map for an element subtree

1. Make sure that Overwrite Mapping mode is selected in the XQuery toolbar.
2. Select the source element. The element can have child elements, or can also be a leaf element.
3. Hold down the left mouse button, and drag the mouse pointer to the target element. Release the left mouse button.

   The source element subtree gets copied to the specified location in the target tree. A solid line connects the root of the copied source subtree to the target subtree. If there are no type mismatches with the target schema, then the copied element and its child elements appear in normal font. If there is a mismatch, the elements show up in underlined font.

How to Use Append Mapping to Copy an Element Subtree to the Target Tree

To create an append map for an element subtree

1. Make sure that Append Mapping mode is selected in the XQuery toolbar.
2. Select the source element. The element can have child elements, or can also be a leaf element.
3. Hold down the left mouse button, and drag the mouse pointer to the target element. Release the left mouse button.

   The source element subtree gets appended as the child of the selected element in the target tree. A solid line connects the root of the source subtree to the root of the appended subtree in the target. If there are no type mismatches with the target schema, then the copied element and its child elements appear in normal font. If there is a mismatch, the elements show up in underlined font.

How to Perform Multiple Value Mappings with One Drag and Drop Action

To create multiple value mapping with one drag and drop action

1. Make sure that Value Mapping mode is selected in the XQuery toolbar.
2. Select the non-leaf source element.
3. Hold down the left mouse button, and drag the mouse pointer to the target element. Release the left mouse button.

   If the source and target elements have the same schema types, individual mappings are created for all the child elements of the source element and target element. A mapping is also created between the source and target element.

   If an element has multiple occurrences, then a FLOWR cycle is automatically created for the element. For example, the code segment below copies each `Item` iteratively:

```
for $Item in $pParam1/Items/Item
            return <Item PartNum="{fn:data($Item/@PartNum)}">
                <ProductName>{fn:data($Item/ProductName)}</ProductName>
                <Quantity>{fn:data($Item/Quantity)}</Quantity>
                <Price>{fn:data($Item/Price)}</Price>
                <Currency>{fn:data($Item/Currency)}</Currency>
                   </Item>
```

Using XQuery Functions

You can add XQuery functions to your existing XQuery map. The Components window contains a list of XQuery functions that you can drag and drop into the source editor or the center pane of the XQuery mapper.

The Components window also includes a set of XQuery constructs, like FLWOR, and XQuery operators, like logical AND. These constructs and operators can only be dragged and dropped into the source editor.

How to Add an XQuery Function in the XQuery Mapper

You can drag and drop an XQuery function from the Components window to the center pane of the XQuery mapper.

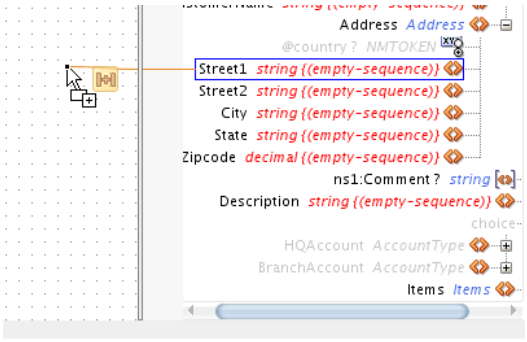
To add an XQuery function:

1. Make sure that the Components Window is visible. The default location is the top right hand corner of Oracle JDeveloper.

   If the Components Window is not visible, select **Components** from the **Window** menu.
2. In the Component Window, select the XQuery Functions page.
3. Click the Category that contains your function. For example, to add the concat function, click String Functions.
4. Drag the desired function from the Components window to the center pane of the XQuery mapper. When you drag the function to the center pane, the output of the function connects to different target nodes, as you move along.

Figure 41-9 shows a function being dragged to the center pane of the XQuery Mapper.

Figure 41-9 Dragging a Function to the Center Pane of the XQuery Mapper



5. Drop the function on the center pane when the function output is shown connected to the desired target node.

Note:

You can also drop a function to an existing map line in the center pane of the XQuery mapper.

The function gets connected to both the source (input) and target (output) nodes.

---

6. If the function requires additional input parameters, then a Warning icon appears on the function icon. Drag a line from a source node to the left end of the function to specify an input parameter.

7. Repeat the previous step for any more source nodes that you must add as input parameters.

To edit a function's parameters:

1. Click the function icon in the center pane. The expression corresponding to the function appears in the Properties window.

   The Properties window is located at the lower right-hand corner of Oracle JDeveloper, by default. If the Properties window does not appear, click Properties under the Window menu to display the Properties window. You can optionally choose to drag the Properties window to any convenient location within the JDeveloper window. You can also resize the Properties window, as desired.

2. Edit the expression that appears in the right pane of the Properties window.

   The left pane of the Properties window shows the variable tree that includes all variables visible in the current scope. You can drag and drop variables to the expression on the right to help build your function definition.

3. Click **Commit** at the top left corner of the Properties window to save the changes. Alternatively, click **Revert to mapper sources** to revert changes made in the Properties window.

Using Library Modules

How to Create an XQuery Main/Library Module discusses the process of creating a library module file. To use a library module, you can import the library module into the main module. This makes all the library module functions available in the main module.

How to Import a Library Module

You can import a library module from the source editor of your main module.

To import a library module:

1. Make sure that your XQuery main module map file is open in the XQuery Mapper.
2. Click the **XQuery Source** tab at the bottom left of the XQuery Mapper window to switch to the source editor.
3. Right-click anywhere in the source editor window. A context menu appears.
4. Select **Import library module** from the context menu. The Select XQuery Library Files dialog appears.
5. Browse and select the XQuery library module file to be imported. Click **OK.**

   An import statement, corresponding to the library module, is added to the main module source view.

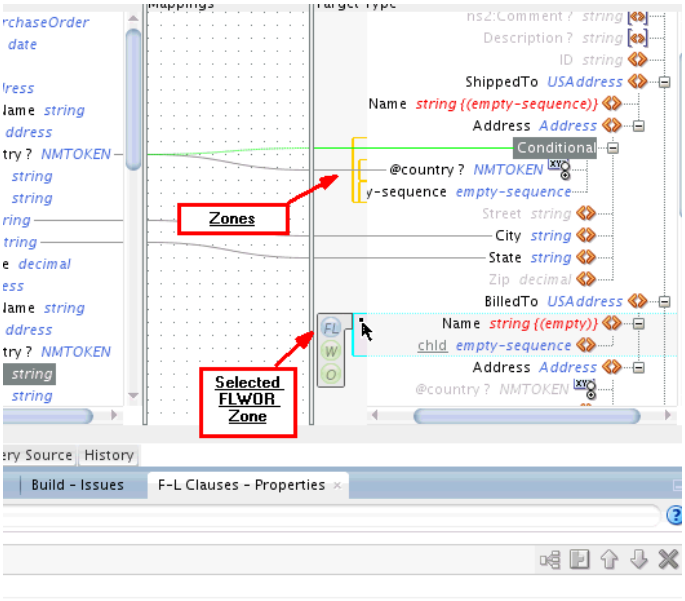
Working with Zones and FLWOR Constructs

You can create FLWOR (For, Let, Where, Order By, Return) expressions in the Source View. FLWOR expressions are represented as zones in the XQuery Mapper target tree.

Zones identify areas in the target tree that are associated with FLOWR constructs or If-Then-Else conditional constructs. Zones are represented by yellow brackets to the left of the target tree.

If you move the mouse over a yellow line representing a zone, the line turns blue. For FLWOR zones, additional buttons appear, corresponding to the For-Let, Where, and Order By clauses. Figure 41-10 shows a sample XQuery with zones.

Figure 41-10 Zones and FLWOR Zones in XQuery Mapper



How to Edit a FLWOR Construct

You can edit a FLWOR construct directly in the source view. You can also perform limited editing of a FLOWR construct in the XQuery Mapper.

To edit a FLWOR construct in XQuery Mapper:

1. Click on the yellow bracket representing the FLWOR zone to select the zone. The yellow bracket turns blue, and additional buttons appear. These buttons correspond to the different clauses of the FLWOR construct.
2. Click the FL button to edit the For-Let properties for the FLWOR construct. The Properties window shows the For-Let clause properties.

   Click the Help icon in the Properties window to display help on editing the For-Let properties.
3. Click the W button to edit the Where properties for the FLWOR construct.

   The Properties window enables you to directly edit the Where expression. You can also drag and drop variables from the left pane of the Properties window.
4. Click the O button to edit the Order By properties for the FLWOR construct.

   The Properties window enables you to directly edit the Order By expression. You can also drag and drop variables from the left pane of the Properties window.

Using Type Annotations to Improve XQuery Performance

When an XQuery is run, the XQuery engine performs schema type validations in the XQuery file before running the XQuery. This may cause performance overheads for certain applications.

If you must optimize your XQuery for performance, you can use type annotations to specify schema information in the XQuery file. Type annotations enable you to hide

the schema type definitions from the XQuery execution engine. The schema definitions are still visible to the Xquery Mapper, which enables you to edit your XQuery map in the usual fashion.

To use type annotations in your XQuery file, select **Use Schema Type Annotations** in the Create XQuery Map Main/Library Module dialog when creating a new XQuery file. See How to Create an XQuery Main/Library Module for more information on creating an XQuery file.

Type annotations, in an XQuery file, look similar to standard XQuery comments. While standard XQuery comments are delimited by the parentheses and colon, type annotations use parentheses and double colons. So, for example:

```
(: This is an XQuery comment :)
(:: This is a type annotation ::)
```

An XQuery file that uses type annotations has the following version annotation at the beginning of the file, immediately following the version declaration:

```
(:: OracleAnnotationVersion "1.0" ::)
```

The following example compares a few XQuery constructs with and without the type annotations.

- Schema import (without type annotation):

  ```
  import schema namespace ns1="http://www.oracle.com/pcbpel/po" at "../Schemas/
  PurchaseOrder.xsd";
  ```

  Schema import (with type annotation):

  ```
  declare namespace ns1="http://www.oracle.com/pcbpel/po";
  (:: import schema at "../Schemas/PurchaseOrder.xsd" ::)
  ```

- Variable declaration (without type annotation):

  ```
  declare variable $test_param as schema-element(ns1:PurchaseOrder) external;
  ```

  Variable declaration (with type annotation):

  ```
  declare variable $test_param as element()
  (:: schema-element(ns1:PurchaseOrder) ::) external;
  ```

Testing Your XQuery Map

You can test run your XQuery map from within Oracle JDeveloper. Testing the XQuery at design time helps prevent runtime errors.

How to Test an XQuery Map

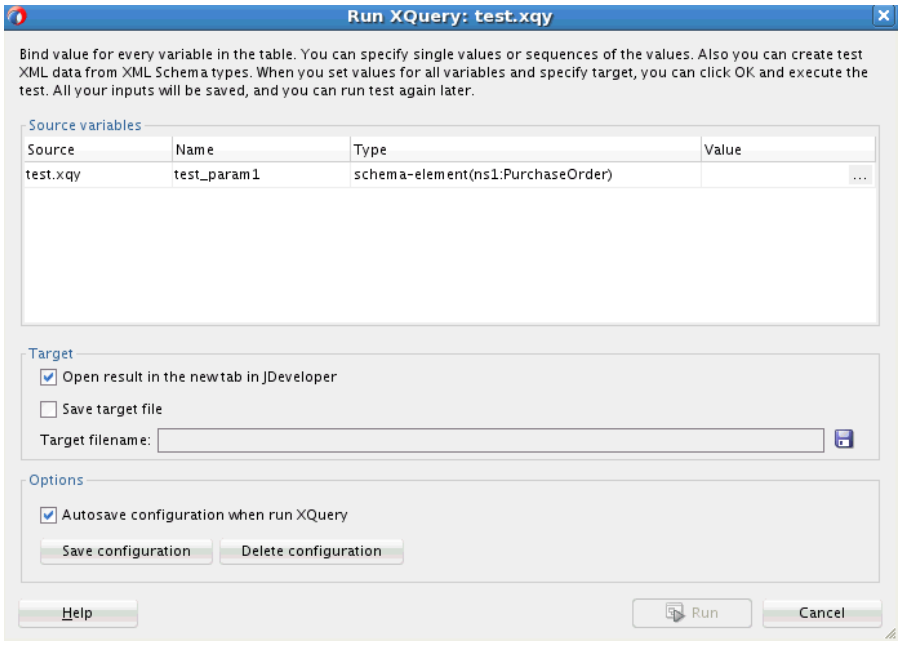
You must be in the Source Editor view to test the XQuery map.

To Test an XQuery Map:

1. Make sure that the XQuery main module is open in the XQuery Mapper.
2. If you are not in the Source Editor view, click the **XQuery Source** tab at the bottom of the XQuery Mapper window to switch to the source editor.
3. Right-click anywhere in the source editor. Select **Run XQuery** from the context menu that appears.

The Run XQuery dialog appears. Figure 41-11 shows the Run XQuery dialog.

Figure 41-11 Run XQuery Dialog



4. Specify values for all source variables that appear in the Source variables section.

   For simple data types, you can specify a value directly under the **Value** field. If your source variable uses a complex schema, click the ellipses (. . .) button to bring up the Edit Variable dialog. You can use an existing XML file to specify test data for the variable, or create an XML file with test data. Click Help for additional help with completing the Edit Variable dialog.

5. Select **Open result in a new tab in JDeveloper** if you want to use a new tab to display test results. Select **Save target file** to save the result file. You must select one, or both, of these options.

6. Click the Save icon to the right of **Target filename** to specify a result file into which the result data is saved.

   You must specify a **Target filename** even if you haven't selected **Save target file** in the preceding step.

7. Optionally select **Autosave configuration** to automatically save the configuration settings when the XQuery is run. The next time you try to run the XQuery, the configuration settings are retrieved.

8. If you want to save the settings made in the Run XQuery dialog, click **Save Configuration.**

9. If you want to delete any previously saved configuration settings, click **Delete Configuration.**

10. Click **Run** to run the XQuery.

42

Using Business Events and the Event Delivery Network

This chapter describes how to subscribe to or publish business events from Oracle Mediator or a BPEL process in a SOA composite application. Business events are published to the Event Delivery Network (EDN) and 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:

- Introduction to Business Events
- Creating Business Events in Oracle JDeveloper
- Subscribing to or Publishing a Business Event from an Service Component
- Subscribing to or Publishing a Business Event from a BPEL Process Service Component
- How to Integrate Oracle ADF Business Component Business Events with Oracle Mediator

For information about creating composite sensors on service components that subscribe to business events, see Defining Composite Sensors .

For information about troubleshooting business events, including specifying the number of threads, stopping event delivery, and specifying the maximum number of deliveries, see Appendix "Troubleshooting Oracle SOA Suite and Oracle BPM Suite" of *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

For information about managing business events from Oracle Enterprise Manager Fusion Middleware Control, see Chapter "Managing Business Events" of *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

For information about business event tuning, see *Tuning Performance*.

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). The EDL is a schema used to build business event definitions. Applications work with instances of the business event definition.

The EDL consists of the following:

- Defined events

  One or more event definitions (`event-definition` element) with the same namespace (`targetNamespace` attribute of `definitions` root element), each having a local name (`name` attribute of the `event-definition` element). The namespace and local name constitute an event name (`QName`).

- Payload definition

  The most common use for a definition is an XML Schema (XSD). The payload of a business event is defined in an XSD that is imported (through the `schema-import` element) into the EDL. Each defined event (that is, `event-definition` element) can have a reference to an imported payload XSD element (the `element` attribute of the `content` element). The schema URI is contained in the root element of the payload.

The following example shows an EDL file with two business events in the `BugReport` event definition: `bugUpdated` and `bugCreated`. The namespace (`/model/events/edl/BugReport`) and associated schema file (`BugReport.xsd`) are referenced.

```
<?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 and a BPEL process.

Business events are deployed to the Oracle Metadata Services Repository (MDS Repository). Deploying a business event to the MDS Repository 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 the MDS Repository. An object in an MDS Repository shared area is visible to all applications in the Resources window of Oracle JDeveloper. After an EDL is published, it can be subscribed to by SOA components such as Oracle Mediator or a BPEL process.

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 to the EDN. The EDN runs within every Oracle SOA Suite 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 is based on a standard JMS messaging infrastructure that supports business event-based interactions among Oracle SOA Suite components and non-Oracle SOA Suite components. The EDN provides two JMS-based types:

- Oracle WebLogic Server JMS: By default, all business events use a single, default Oracle WebLogic Server JMS topic.
- Oracle Advanced Queueing (AQ) JMS

You can create additional JMS topics (Oracle WebLogic Server JMS or AQ JMS) and map different event types to these additional JMS topics in Oracle Enterprise Manager Fusion Middleware Control.

Note:

- The EDN does not support durable subscriptions (whether they are backed by native AQ or Oracle WebLogic Server JMS). The subscribing service component must be running to receive events.

EDN Integration with Oracle SOA Suite

Oracle SOA Suite EDN provides the following features:

- A standard JMS-based messaging infrastructure that provides the following:
  - A JMS-based event publish and subscribe architecture for Oracle SOA Suite and non-Oracle SOA Suite clients.
  - Support for bidirectional interactions (can both publish to and subscribe from Oracle SOA Suite and non-Oracle SOA Suite clients).
  - Support for both the Oracle AQ JMS and Oracle WebLogic Server JMS providers. An Oracle WebLogic Server JMS topic (default) and an AQ JMS topic are automatically configured for EDN use after installation. The default JMS type can be switched from Oracle WebLogic Server JMS (default) to AQ JMS in

Oracle Enterprise Manager Fusion Middleware Control. For more information, see "Mapping Business Events to JMS Topic Destinations on the Business Events Page" of *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

  - EDN support as a lightweight manager above both JMS providers.
  - A plain JMS API and an Oracle SOA Suite, value-added EDN API for remote, non-Oracle SOA Suite clients to use for integrating with Oracle SOA Suite. For more information, see *Java API Reference for Oracle SOA Suite Event Delivery Network*.
  - JMS transactions to guarantee EDN delivery (for both the one-and-only-one (OAOO) and guaranteed consistency methods).
  - Durable and persistent publishing delivery options to prevent message loss. These default options are beneficial for interactions with remote, non-Oracle SOA Suite clients.
  - A JMS adapter used internally for implementing many JMS features. For information about the JMS adapter, see the "Oracle JCA Adapter for JMS" chapter of *Understanding Technology Adapters*.
  - No duplicate event processing in a multinode cluster.

- Scalability at a fine-grained level. This enables different events to map to different JMS topic destinations, thereby eliminating the need for a single location to handle all events. This reduces potential bottlenecks. Mapping is performed by an administrator in Oracle Enterprise Manager Fusion Middleware Control. For more information, see the "Mapping Business Events to JMS Topic Destinations" section of *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.
- Support for the following publish and subscribe scenarios:
  - Publish and subscribe to events across the same composite or different composites.
    - Use the default EDN Oracle WebLogic Server JMS topic automatically provided.
    - Use the custom event-to-JMS-topic mapping provided in Oracle Enterprise Manager Fusion Middleware Control.
  - Publish and subscribe to events with remote, non-Oracle SOA Suite participants through one of the following APIs:
    - Plain JMS API (for J2SE client environments)
    - EDI API `EdnJmsConnection` (for J2SE and J2EE client environments)
- Instance tracking and fault recovery support in the Error Hospital. For more information, see *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.
- The storage of EDL files in the MDS Repository. This makes the files available for browsing in the Resources window in Oracle JDeveloper. For more information, see Managing Shared Data with the Design-Time .

Note:

For memory recommendations on sending large payloads in the event delivery network (EDN) with Oracle AQ JMS, see JVM Memory Sizing Recommendations for SOA Composite Applications.

Business Event API Support for Remote Clients

For remote clients to publish and subscribe to events in Oracle SOA Suite, there are several API options. Table 42-1 provides details.

Table 42-1 Remote API Options

Option	Description	Supported By	Advantages/Disadvantages
Plain JMS API	Use to directly interact with EDN JMS topics. This is typically a J2SE client with raw JMS access. The remote client must configure JNDI properties to point to the SOA server.	• Oracle WebLogic Server JMS • AQ JMS	The advantages are: • Supports the standard JMS API, meaning you can use many JMS software tools. The disadvantages are: • Service level degradation. • Requires manual discovery of mapped JMS and configuration of JNDI. • Requires extra coding, including handling of the internal EDN event structure, filter translation, subject propagation, transaction, error handling, and so on.
EDN API - `EdnJmsConnection`	For a J2SE client, such as Oracle Event Processing. This option provides all standard publish and subscribe options. The remote client must perform the following tasks: • Configure JNDI properties to point to the SOA server. • Invoke the EDN helper method `findRelevantBEConnFactory` to return an appropriate connection factory. This enables you to use a JMS connection for publishing and subscribing to events.	• Oracle WebLogic Server JMS • AQ JMS	The advantages are: • No client JNDI configuration or JMS adapter deployment • Handles JMS mapping, conversion, and translation. The disadvantages are: • Based on the plain JMS connection factory and topic. For information about the JMS adapter, see the "Oracle JCA Adapter for JMS" chapter of *Understanding Technology Adapters*.

For more information about the EDN APIs, see *Java API Reference for Oracle SOA Suite Event Delivery Network*.

Guidelines for Manually Setting Event Delivery Network Properties When Invoking the BusinessEvent.setProperty API

When publishing an event delivery network (EDN) business event, most properties cannot be manually set by invoking the `BusinessEvent.setProperty(String name, Object value)` API.

Properties That Cannot Be Manually Set

Do not set the following EDN business event properties. The values for these properties are internally set and used by EDN.

- General properties:
  - `BusinessEvent.EVENT_ID ("id")`
  - `BusinessEvent.PARENT_ID ("parent-id")`
  - `BusinessEvent.PUBLISHED_TIME ("published-time")`
  - `BusinessEvent.OWNER ("owner")`
  - `BusinessEvent.SOURCE ("source")`
  - `BusinessEvent.MODE ("mode")`
- *All* tracking properties, for example:
  - `BusinessEvent.PROPERTY_ECID ("tracking.ecid")`
  - `BusinessEvent.PROPERTY_COMPOSITE_INSTANCE_ID ("tracking.compositeInstanceId")`
  - `BusinessEvent.PROPERTY_PARENT_COMPONENT_INSTANCE_ID ("tracking.parentComponentInstanceId")`
  - `BusinessEvent.PROPERTY_CONVERSATION_ID ("tracking.conversationId")`
  - `tracking.compositeInstanceCreatedTime"`

Properties That Can Be Manually Set

You can set the following properties:

- `BusinessEvent.PRIORITY ("priority")`
- `BusinessEvent.CONTEXT ("context")`

Local and Remote Event Connections

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.

- 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.

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.

Creating Business Events in Oracle JDeveloper

This section provides a high-level overview of how to create and subscribe to a business event.

How to Create a Business Event

To create a business event:

1. Create a SOA project as an empty composite.
2. Launch the Create Event Definition wizard in either of the following ways:
   a. From the **File** main menu, select **New** > **Event Definition**.
   b. From the **File** main menu, select **New** > **Application** > **SOA Tier** > **Service Components** > **Event Definition**.

   The Create Event Definition dialog appears.
3. Enter the details described in Table 42-2.

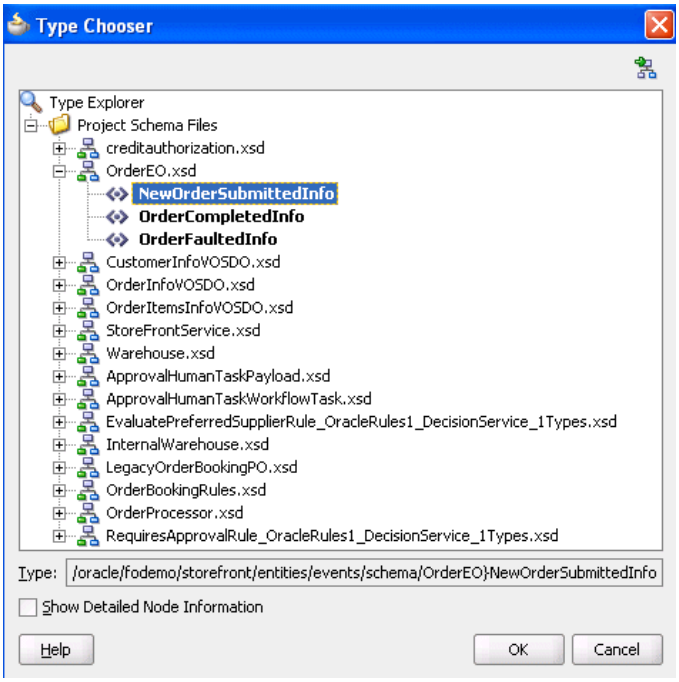
Table 42-2 Create Event Definition Dialog Fields and Values

Field	Value
Name	Enter a name or accept the default name of `EventDefinition`*`number`*. The name you enter here becomes the EDL file name in the Applications window. **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 in which to create the event definition file.
Namespace	Accept the default namespace or enter a value for the namespace in which to place the event. This enables the subscriber to receives events of the indicated namespace.

4. Click the **Add** icon to add an event.

   The Create Event dialog appears.
5. Click the **Search** icon to select the payload, and click **OK**. Figure 42-1 provides details.

Figure 42-1 Select the Payload



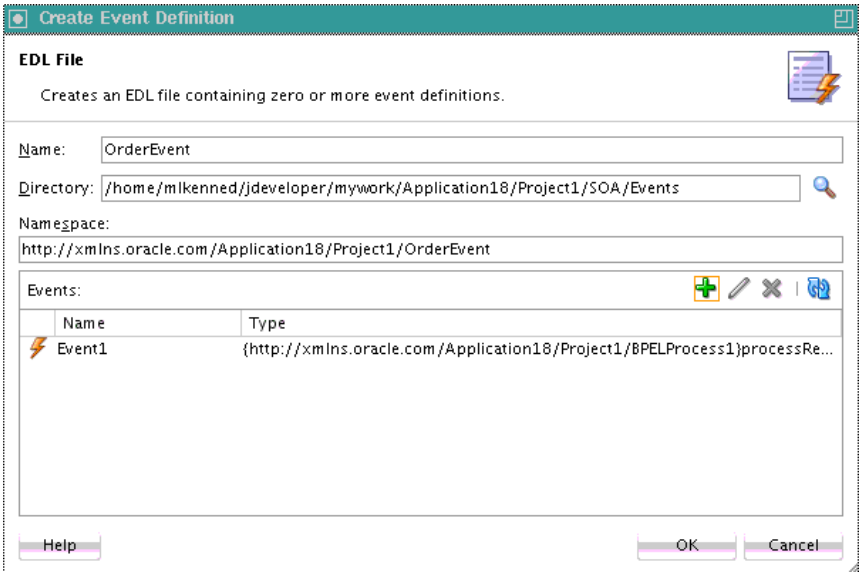
You are returned to the Create Event dialog.

6. In the **Name** field, enter a name.

7. Click **OK**.

   The added event now appears in the **Events** section. Figure 42-2 provides details.

Figure 42-2 Create Event Definition



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 the MDS Repository and you are returned to the . The business event displays for browsing in the Resources window.

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.

How to Subscribe to a Business Event

To subscribe to a business event:

1. From the Components window, drag a **Mediator** service component into the . This service component enables you to subscribe to the business event.
2. In the **Name** field, enter a name.
3. From the **Template** list, select **Subscribe to Events**.

   The dialog is refreshed to display an events table.
4. Click the **Add** icon to select an event.

   The Event Chooser dialog appears.
5. Select an existing event or click the **Add** icon to create a new event, and click **OK**.

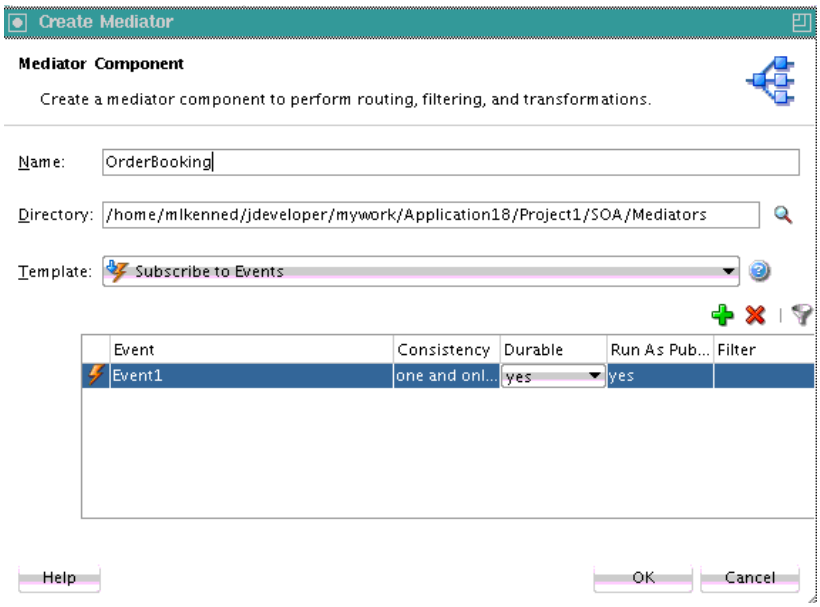
   You are returned to the Create Mediator dialog.
6. Complete the remaining fields of the dialog. Table 42-3 provides details.

Table 42-3 Events Table of Create Mediator Dialog

Element	Description
Consistency	Click inside the **Consistency** column to 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 with retriable exceptions are automatically retried a configured number of times before they are moved to the Error Hospital for recovery (that is, subject to manual retries). Failed events with nonretriable exceptions are moved to the Error Hospital without automatic retries, and are not recoverable. • **guaranteed** Events are delivered to the subscriber in a local JMS transaction. The subscriber can choose to create its own local transaction for processing, but it is committed independently of the rest of event processing. The **guaranteed** consistency level is a lower quality of service option than **one and only one**, because a local transaction is used instead of a global transaction. Failed events with retriable exceptions are automatically retried a configured number of times before they are moved to the Error Hospital where they are recoverable, (that is, subject to manual retries. Failed events with nonretriable exceptions are moved to the Error Hospital without automatic retries, and are not recoverable. For information about the Error Hospital, see Section "Recovering From Faults in the Error Hospital" of *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.
Durable	Durable subscriptions prevent against message loss caused by different life cycles of publishers, subscribers, and the framework. Select an option: • **yes**: Events are retained if the subscriber is not running. This is the default selection. • **no**: Events are dropped if the subscriber is not running.
Run as Publisher	Select a security publishing option: • **yes**: The subscriber has the event publisher's security identity. This is the default selection. • **no**: The subscriber does not have the event publisher's security identity.
Filter	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 How to Specify an Expression for Filtering Messages.

Figure 42-3 shows the Create Mediator dialog.

Figure 42-3 Create Mediator Dialog



7. Click **OK**.

   Figure 42-4 shows an icon on the left side that indicates that Oracle Mediator is configured for an event subscription.

Figure 42-4 Configuration for Event Subscription

How to Publish a Business Event

You can create a second Oracle Mediator to publish the event that you subscribed to in How to Subscribe to a Business Event. While not shown here, you can also create a BPEL component to publish the 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**.

What Happens When You Create and Subscribe to a Business Event

The source code in the following example provides details about the subscribed event of the Oracle Mediator service 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"
         durable="true" runAsRoles="$publisher"/>
</business-events>
</component>
```

While not explicitly demonstrated in this example, you can define XPath filters on events. In the following example, the event is accepted for delivery *only* if the initial deposit is greater than `50000`:

```
    <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>
```

What Happens When You Publish a Business Event

Two Oracle Mediator service components appear in the following example. One service component (`OrderPendingEvent`) subscribes to the event and the other service component (`PublishOrderPendingEvent`) publishes the event.

```
<component name="PublishOrderPendingEvent">
    <implementation.mediator src="PublishOrderPendingEvent.mplan"/>
    <business-events>
      <publishes xmlns:sub1="/oracle/fodemo/storefront/entities/events/edl/OrderEO"
 name="pub1:NewOrderSubmitted" persistent="true" priority="7"
 timeToLive="36000000"/>
    </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"
          durable="true" runAsRoles="$publisher"/>
</business-events>
</component>
```

What You May Need to Know About Subscribing to a Business Event

Only subscribers in default revisions of the SOA composite applications can receive business events. For example, note the following behavior.

To subscribe to a business event:

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.

   Oracle Mediator M2 writes the output to one file in the directory. As expected, Oracle Mediator M3 picks up the event and writes the output successfully to another directory. However, Oracle Mediator M2 (from revision 1) is not picking up the published event from revision 2 of the composite application.

What You May Need to Know About Publishing Events Across Domains Using SAF

When publishing events across domains using Store-and-Forward (SAF), local subscribers cannot subscribe to the event. For example, assume you have the following subscribers:

- Local subscriber (deployed on the same domain as the event publisher)
- Remote subscriber (deployed on a domain external to the event publisher)

Both subscribe to the same event (for this example, named E), which has been configured to listen to the SAF topic. In this environment, only the remote subscriber can subscribe to the event. The local subscriber cannot subscribe to the event.

The JMS topic for EDN must be provisioned as a physical JMS topic instead of as an imported SAF topic. This is because an imported SAF topic has its own rules of context lookup and security checking that EDN does not natively support.

Workaround for Local Subscribers

As a workaround, you must perform the following procedures:

1. Create a local JMS topic that the publisher can locate. For example, in local domain A, which the event publisher can locate, you provision a regular Oracle WebLogic Server JMS topic (for example, named Ta) to which to publish events, and a subscriber (local in domain A) to listen for this topic.
2. In remote domain B, which a remote subscriber can locate, you create an imported SAF topic (for example, named safTb) that maps to topic Ta from domain A, and have the remote subscriber listen to safTb.

As an alternative to Step 2, you can provision another JMS topic (for example, named Tb) in domain B to which a remote subscriber listens, and create a JMS bridge that bridges source topic Ta to destination topic Tb.

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** check box, and click **Finish**.
7. In the **Name** column, click the provider name you entered in Step 5.
8. Enter the details shown in Table 42-4, and click **Save**.

Table 42-4 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 42-5, and click **OK**.

Table 42-5 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 42-6, and click **OK**.

Table 42-6 Configuration Details

Field	Description
Name	Enter `SOA_EDNLocalTxDataSource`.
Local Name	Enter `jdbc/EDNLocalTxDataSource`.
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.

How to Configure the Connection Factory When the Oracle WebLogic Server JMS Runs in the Same Local JVM as the JMS Adapter

If Oracle WebLogic Server JMS is running in the local JVM (the same JVM as the JMS adapter), you must correctly configure the `isTransacted` connector factory property. For your servlet client, which is locally colocated with the Oracle WebLogic Server JMS server to work, perform the following steps:

1. Log in to Oracle WebLogic Server Administration Console, and select **Deployments** > **JmsAdapter** > **Configuration** -> **Outbound Connection Pools**.
2. Expand groups and instances, and select both **eis/wls/EDNLocalTxDurableTopic** and **eis/wls/EDNLocalTxTopic**.
3. Set **isTransacted** to **false**.
4. Save and restart the SOA server.

   For more information, see Section "Synchronous/Asynchronous Request Reply Interaction Pattern" of *Understanding Technology Adapters*.

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.

How to Subscribe to a Business Event

To subscribe to a business event:

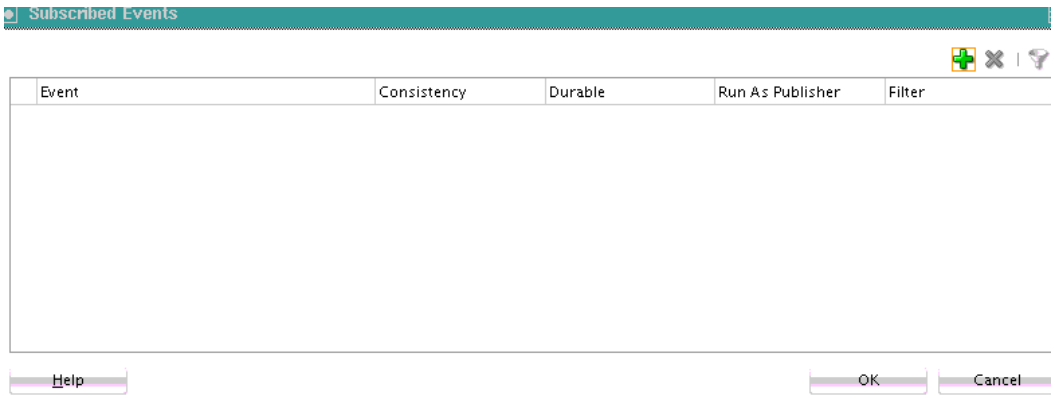
1. From the Components window, drag a **BPEL Process** service component into the .
2. In the **Name** field, enter a name. Do not change any other default option and click **OK**.

   The BPEL process service component is created.
3. Double-click the BPEL process service component. 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 Components window 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 Selecting Between Continuing or Waiting on a Process with a Pick Activity.
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 42-5.

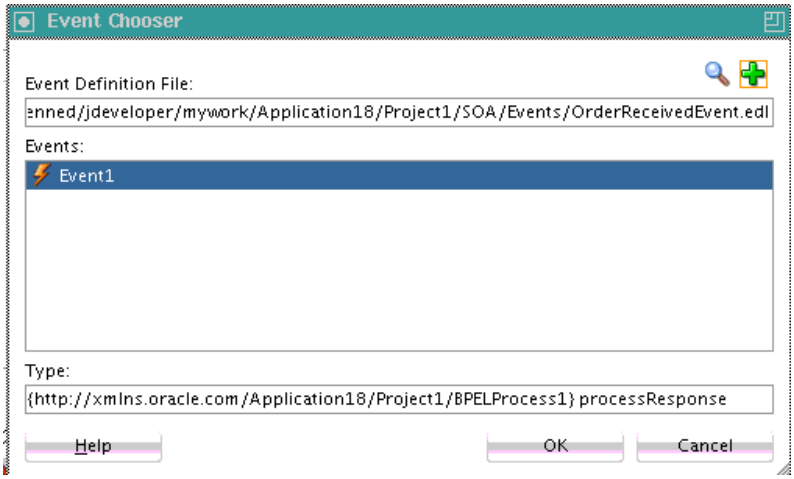
   Figure 42-5 Subscribed Events Dialog

9. Click the **Add** icon to select an event.

   The Event Chooser dialog appears, as shown in Figure 42-6.

Figure 42-6 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. Table 42-7 provides details.

Table 42-7 Events Table of Subscribed Events Dialog

Element	Description
Consistency	Click inside the **Consistency** column to 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 with retriable exceptions are automatically retried a configured number of times before they are moved to the Error Hospital for recovery (that is, subject to manual retries). Failed events with nonretriable exceptions are moved to the Error Hospital without automatic retries, and are not recoverable. • **guaranteed** Events are delivered to the subscriber in a local JMS transaction. The subscriber can choose to create its own local transaction for processing, but it is committed independently of the rest of event processing. The **guaranteed** consistency level is a lower quality of service option than **one and only one**, because a local transaction is used instead of a global transaction. Failed events with retriable exceptions are automatically retried a configured number of times before they are moved to the Error Hospital where they are recoverable, (that is, subject to manual retries. Failed events with nonretriable exceptions are moved to the Error Hospital without automatic retries, and are not recoverable. For information about the Error Hospital, see Section "Recovering From Faults in the Error Hospital" of *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

Table 42-7 (Cont.) Events Table of Subscribed Events Dialog

Element	Description
Durable	Durable subscriptions prevent against message loss caused by different life cycles of publishers, subscribers, and the framework. Select an option: • **yes**: Events are retained if the subscriber is not running. This is the default selection. • **no**: Events are dropped if the subscriber is not running.
Run as Publisher	Select a security publishing option: • **yes**: The subscriber has the event publisher's security identity. This is the default selection. • **no**: The subscriber does not have the event publisher's security identity.
Filter	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 How to Specify an Expression for Filtering Messages.

12. Click **OK** to close the Subscribed Events dialog.

    You are returned to the Receive dialog.

    Note:

    Optionally, you can select the **Create Instance** check box, 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.

13. Click **OK**.

    Figure 42-7 shows a BPEL process service component that is configured for event subscription.

Figure 42-7 BPEL Process Service Component Configuration for Event Subscription

How to Publish a Business Event

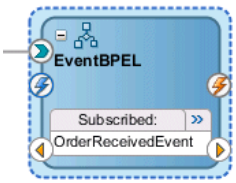
To publish a business event:

1. Drag an **Invoke** activity from the Components window into the SOA Composite Editor, below the **Receive** activity created in 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. To the right of the **Event** field, click the **Browse Events** icon. The Event Chooser dialog appears.
6. Select the event you created and click **OK**.

   You are returned to the Invoke dialog.
7. Click **OK**.

   Figure 42-8 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 42-8 BPEL Process Service Component Configuration for Event Subscription and Publishing



What Happens When You Subscribe to and Publish a Business Event

The source code in the following example shows how the `composite.xml` source changes for the subscribed and published events of a BPEL process service component.

```
<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"
                durable="true" runAsRoles="$publisher"/>
     <publishes xmlns:pub1="http://mycompany.com/events/orders"
                name="pub1:ProductSoldAlert" persistent="true" priority="7"
                timeToLive="36000000"/>/>
   </business-events>
</component>

    <business-events>
      <publishes xmlns:sub1="/oracle/fodemo/storefront/entities/events/edl/OrderEO"
 name="pub1:NewOrderSubmitted" persistent="true" priority="7"
 timeToLive="36000000"/>
    </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 the following example, the event is accepted for delivery *only* if the initial deposit is greater than `50000`.

```
    <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 the following example:

```
<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>
```

The following example shows how the `eventName` attribute is used in the BPEL source file:

```
<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) activity, then the standard attributes such as `partnerLink`, `operation`, or `portType` 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 `portType` attributes are no longer mandatory. The Oracle JDeveloper validation logic enforces the presence of either the `bpelx:eventName` attribute or the `partnerLink`, `operation`, and `portType` attributes, but not both. The following example shows the modified 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.

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 Oracle 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 *Developing Fusion Web Applications with 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 an Oracle Mediator service component as described in How to Subscribe to a Business Event.

6. In the Event Chooser window, select the EDL file of the event, as described in How to Subscribe to a Business Event.

7. Create a BPEL process service component in the same SOA composite application for Oracle Mediator to invoke. 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 .

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.

43

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. It includes a reference of the XRef functions you can use to populate, view, and maintain entries in the cross reference tables.

This chapter includes the following sections:

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 Working with Domain Value Maps and Using 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 an 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.

Introduction to Cross Reference Tables

Cross references are stored in the form of tables. Table 43-1 shows a cross reference table containing information about customer identifiers in different applications.

Table 43-1 Cross Reference Table Sample

SAP	EBS	SBL
SAP_001	EBS_1001	SBL001
SAP_002	EBS_1002	SBL002

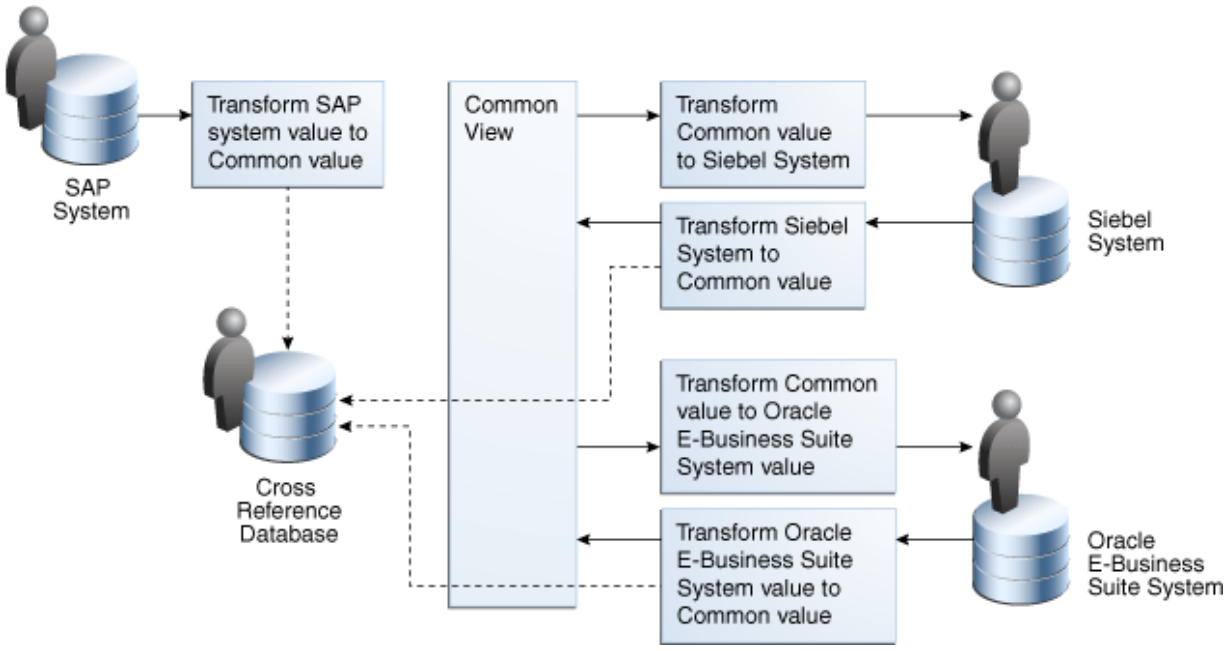
The identifier mapping is also required when information about a customer is updated in one application and the changes must be propagated to other applications. You can integrate different identifiers by using a common value integration pattern, which maps to all identifiers in a cross reference table. For example, you can add one more column named Common to the cross reference table shown in Table 43-1. The updated cross reference table then appears, as shown in Table 43-2.

Table 43-2 Cross Reference Table with Common Column

SAP	EBS	SBL	Common
SAP_001	EBS_1001	SBL001	CM001
SAP_002	EBS_1002	SBL002	CM002

Figure 43-1 shows how you can use common value integration patterns to map identifiers in different applications.

Figure 43-1 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 43-3
- `CUSTOMER` with cross reference columns `EBS`, `COMMON`, and `PORTAL`, as shown in Table 43-4

Table 43-3 ORDER Table

Column Name	SIEBEL	COMMON	EBS
`Column Value`	`SBL_101`	`COM_100`	`EBS_002`
`Column Value`		`COM_110`	`EBS_012`

Table 43-4 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 43-5.

Table 43-5 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 43-6 and Table 43-7.

Table 43-6 XREF_ORDER Table

ROW_ID	SIEBEL	COMMON	EBS
100012345	SBL_101	COM_100	EBS_002
110012345		COM_110	EBS_012

Table 43-7 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 How to Create Custom Database Tables.

Oracle Data Integrator Support for Cross Referencing

Oracle Data Integrator (ODI) achieves data integration through an E-LT (extract, load, transform) model. You can use ODI to help with your cross-referencing needs. ODI provides three Knowledge Modules for handling SOA cross references that perform the following functions: Populate the cross-reference table, create a common ID for the target table, push the common ID and the source primary key to the cross-reference table, and create and push a unique row number that creates the cross reference between the source primary key and the common ID. With the modules, you can create an integration interface that both loads a target table from several source tables and handles cross-references between one of the sources and the target.

For more information about ODI and cross referencing, see "Oracle SOA Suite Cross References" in *Oracle Fusion Middleware Connectivity and Knowledge Modules Guide for Oracle Data Integrator*.

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.

Note:

You can also create cross-reference tables in Service Bus projects and use them in message flows during transformations.

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:

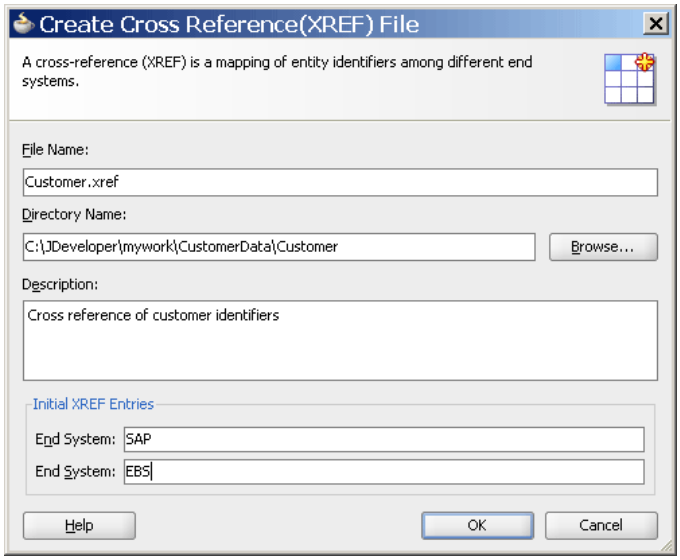
   ```
   Cross reference of Customer identifiers.
   ```

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 43-2.

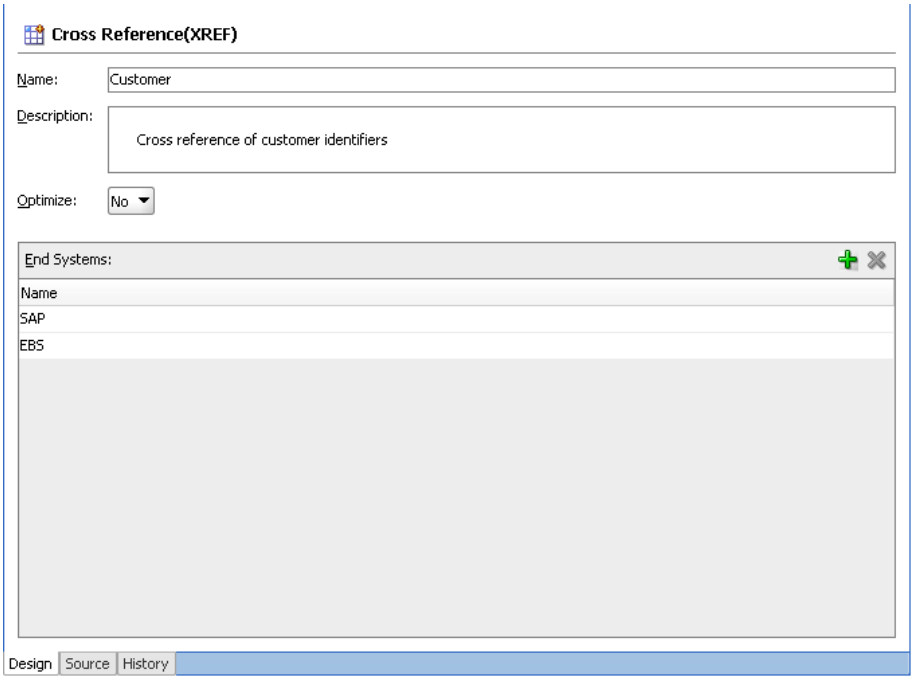
Figure 43-2 Create Cross Reference(XREF) File Dialog



9. Click **OK**.

   The Cross Reference Editor is displayed, as shown in Figure 43-3. You can use this editor to modify the cross reference.

Figure 43-3 Cross Reference Editor



What Happens When You Create a Cross Reference

A file with extension `.xref` gets created and appears in the Applications window. All `.xref` files are based on the schema definition (XSD) file shown in the following example:

```
<?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>
```

How to Create Custom Database Tables

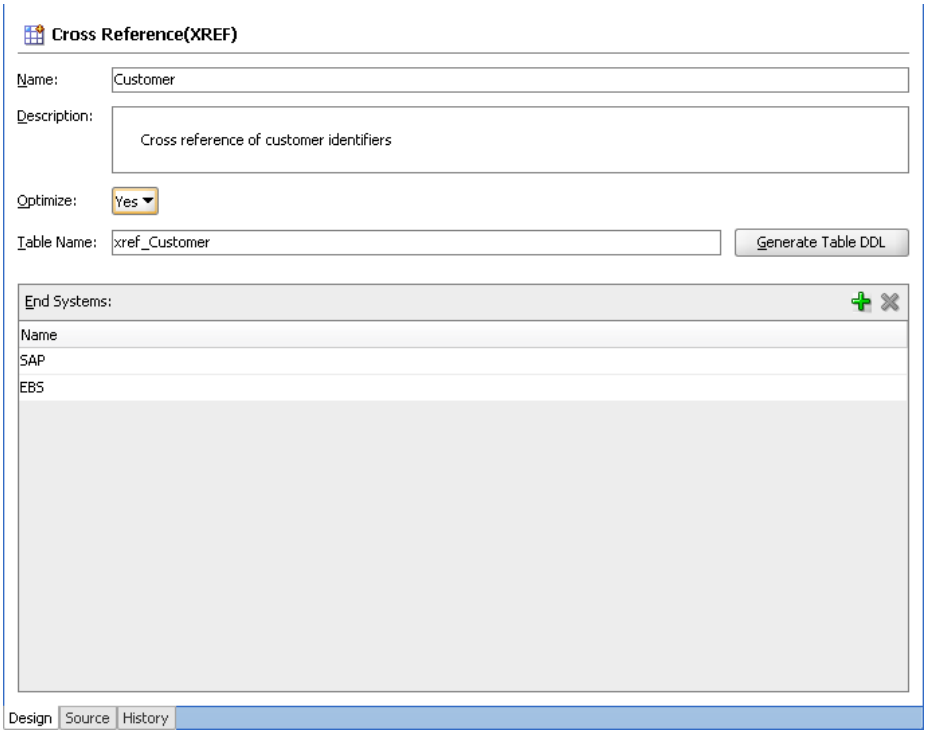
As mentioned previously, all the runtime data by default gets stored in the XREF_DATA table. If you want to create custom database tables, then perform the following steps.

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 43-4.

Figure 43-4 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** check box 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:

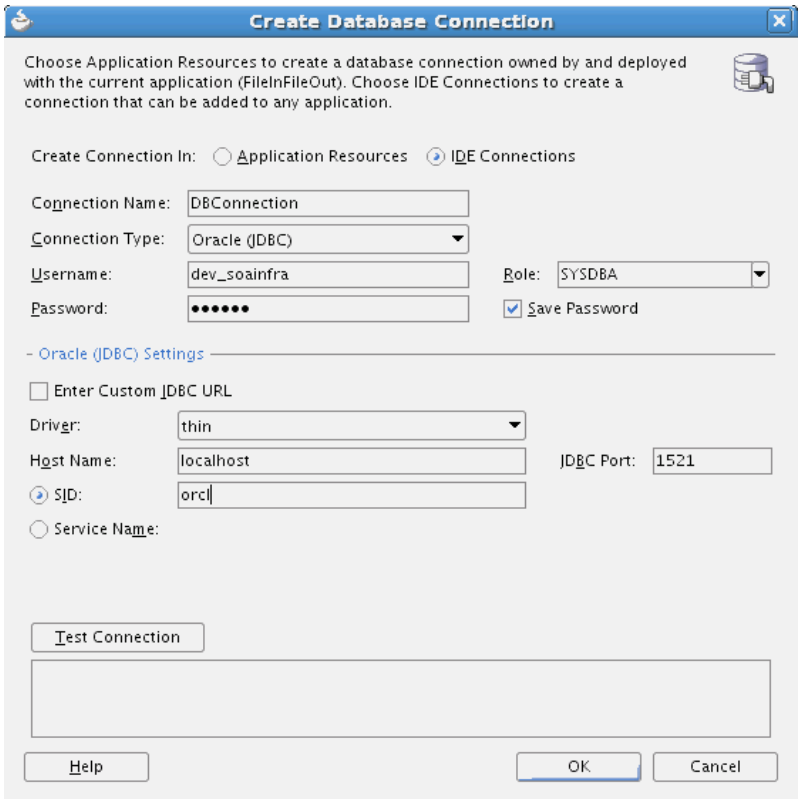
   ```
   Running the Drop DDL will remove the table and indexes, do you want to
   continue?
   ```

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 43-5. 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 43-5 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.

Note:

Create the database table in the soainfra schema of the database.

---

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 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"/>
```

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.

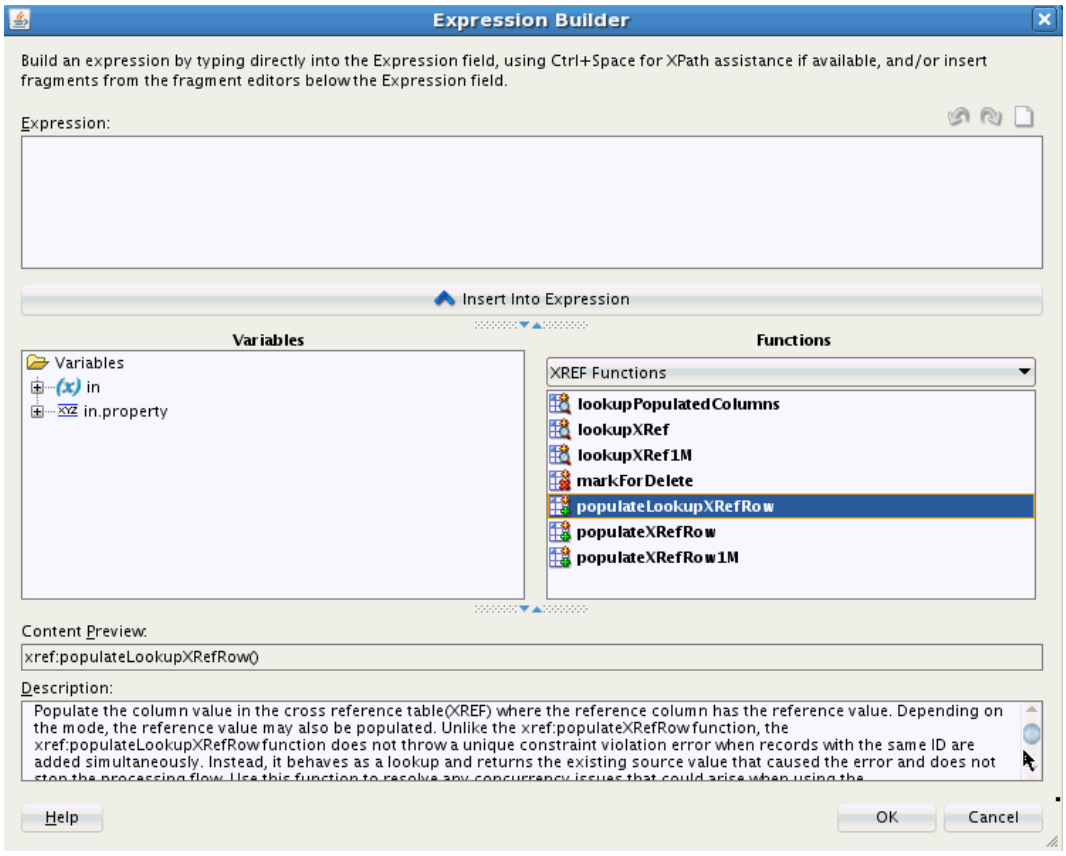
3. Enter the end system name. For example, `SBL`.

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.

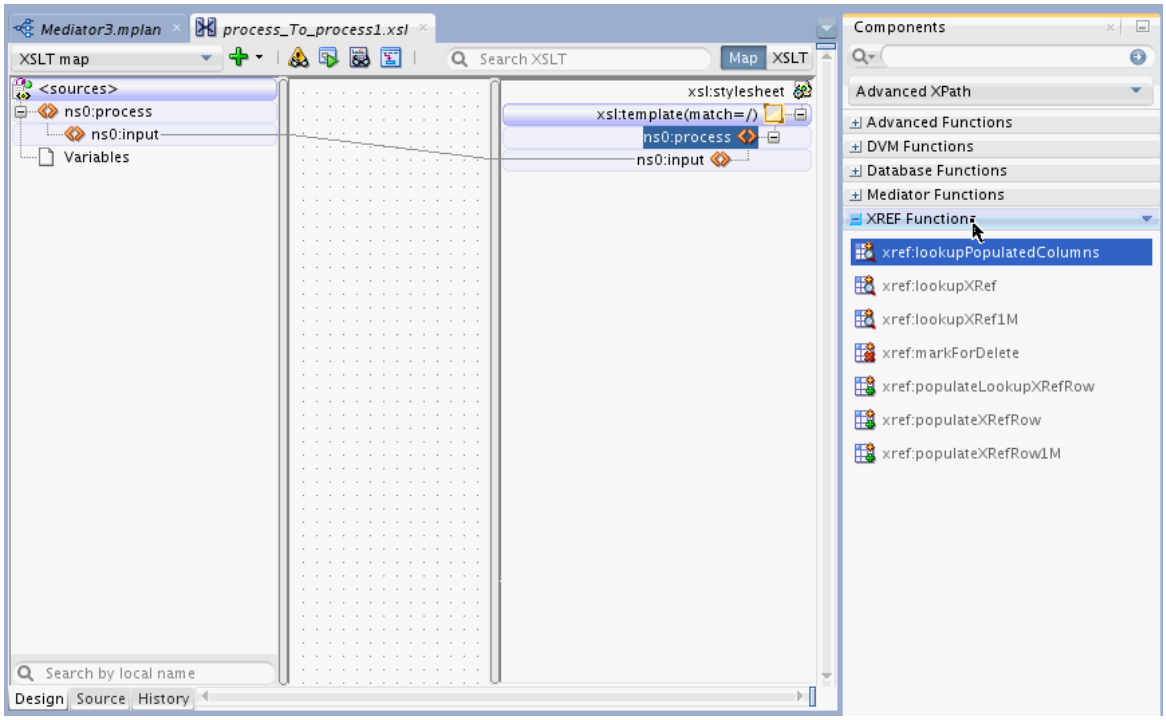
You can access the Expression Builder dialog through an assign activity, an XSL transformation, or the filtering functionality of a BPEL process service component or an Oracle Mediator service component. Figure 43-6 shows how you can select the cross reference functions in the Expression Builder dialog.

Figure 43-6 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 43-7 shows how you can select the cross reference functions in the XSLT Mapper.

Figure 43-7 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.

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 43-8, the `Order` table has the following columns: `EBS`, `Common`, and `SBL` with values `E100`, `100`, and `SBL_001` respectively.

Table 43-8 Cross Reference Table with Single Column Values

EBS	Common	SBL
E100	100	SBL_001

Note:

If you find you have concurrency issues when using this function, you can also use the `populateLookupXRefRow` function. The `populateLookupXRefRow` function should only be used in cases where simultaneous updates are being made, resulting in unique constraint violations. This function is described under About the xref:populateLookupXRefRow Function.

The syntax of the `xref:populateXRefRow` function is shown in the following example:

```
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 43-9 describes these modes.

Table 43-9 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.

Table 43-9 (Cont.) xref:populateXRefRow Function Modes

Mode	Description	Exception Reasons
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 43-9.

Table 43-10 describes the `xref:populateXRefRow` function modes and exception conditions for these modes.

Table 43-10 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

About the xref:populateLookupXRefRow Function

Like the `xref:populateXRefRow` function, the `xref:populateLookupXRefRow` function populates a cross reference column with a single value. Unlike the `xref:populateXRefRow` function, the `xref:populateLookupXRefRow` function does not throw a unique constraint violation error when records with the same ID are added simultaneously. Instead, it behaves as a lookup and returns the existing source value that caused the error and does not stop the processing flow. Use this function to resolve any concurrency issues that could arise when using the `xref:populateXRefRow` function.

The `xref:populateLookupXRefRow` function returns a string value, which is the cross reference value being populated or, with a unique constraint violation, the cross reference value that was already populated by the first committed thread. For example, as shown in Table 43-8, the `XREF_CUSTOMER_DATA` table has the following columns: `EBS`, `Common`, and `SBL`. The `xref:populateLookupXRefRow` function is invoked by two threads in parallel with following values:

- Thread One: `xref: populateLookupXRefRow ("default/xref/example.xref", "EBS", "EBS100", "Common" "CM001", "ADD")`
- Thread Two: `xref: populateLookupXRefRow ("default/xref/example.xref", "EBS", "EBS100", "Common" "CM002", "ADD")`

The table is populated as shown in Table 43-11. Since thread one is committed first, thread two returns "CM001" to the caller.

Table 43-11 Cross Reference Table Populated by xref:populateLookupXRefRow

EBS	Common	SBL
EBS100	CM001	

The syntax of the `xref:populateLookupXRefRow` function is shown in the following example:

```
xref:populateLookupXRefRow(xrefMetadataURI as string, xrefReferenceColumnName as
 string, xrefReferenceValue as string, xrefColumnName as string, xrefValue as
 string, mode as string) as string
```

Parameters

- `xrefMetadataURI`: 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 `ADD` or `LINK`. Table 43-10 describes these modes and exception conditions for the modes.

Note:

The mode parameter values are case-sensitive and should be specified in upper case only.

Table 43-12 xref:populateLookupXRefRow Function Results with Different Modes

Mode	Reference Value	Value to be Added	Result
ADD	Absent	Absent	Success
	Present	Absent	Exception (Success only when Exception is Unique constraint violation)
	Present	Present	Exception (Success only when Exception is Unique constraint violation)
LINK	Absent	Absent	Exception
	Present	Absent	Success
	Present	Present	Exception

Usage Notes

- When using a custom table approach, you must add the primary constraint on the columns that must be unique in the cross-reference table. Using Table 43-11 as an example, the SQL statement is similar to the following:

  ```
  alter table xref_customer_data add constraint xref_vnx_data_pk
   primary key (common, ebs);
  ```

  Populate the primary constraint columns first and then populate the remaining columns in subsequent calls.

- This function should not be used for inserting cross references for primary objects, since this could mask data inconsistency issues. Only use the function for secondary objects to a main dependent object. For example, do not use the function to determine whether an account already exists when creating customer accounts; but do use it if the addresses in those customer accounts are being synchronized.

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 43-13, 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 43-13 Cross Reference Table with Multiple Column Values

SAP	EBS	SBL
SAP_001 SAP_0011	EBS_1001	SBL001
SAP_002	EBS_1002	SBL002

The syntax of the `xref:populateXRefRow1M` function is shown in the following example:

```
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 43-14 describes these modes:

Table 43-14 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.

Table 43-14 (Cont.) xref:populateXRefRow1M Function Modes

Mode	Description	Exception Reasons
`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 43-15 describes the `xref:populateXRefRow1M` function modes and exception conditions for these modes.

Table 43-15 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

How to Populate a Column of a Cross Reference Table

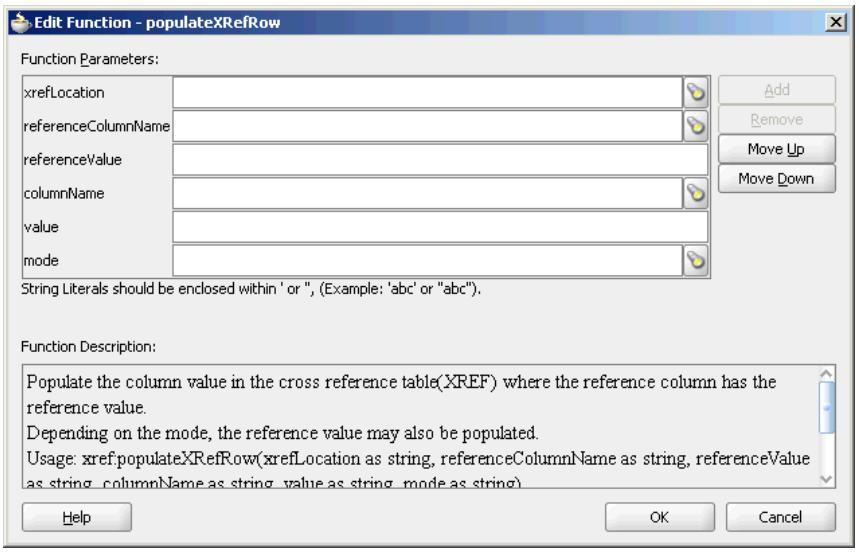
To populate a column of a cross reference table:

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 Components window, 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 43-8.

Figure 43-8 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.

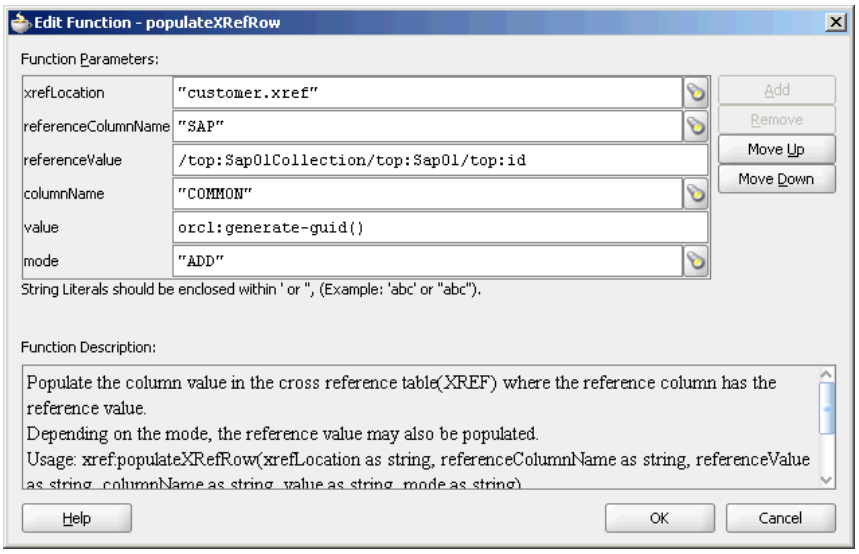
   f. In the **mode** field, enter a mode in which you want to populate the cross reference table column. For example, enter `ADD`.

      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 43-9.

Figure 43-9 Populated Edit Function – populateXRefRow Dialog



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.

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 43-2 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 the following example:

```
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.

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 43-13 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 the following example:

```
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 43-16 with the following three columns: `Siebel`, `Billing1`, and `Billing2`.

Table 43-16 Order Table

Siebel	Billing1	Billing2
100	101	102
110		111
		112

For 1:1 mapping, the `xref:lookupPopulatedColumns("Order","Siebel","100","false")` method returns the values shown in the following example:

```
<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 the following example:

```
<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.

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 the following example:

```
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.

How to Look Up a Cross Reference Table for a Value

To look up a cross reference table column:

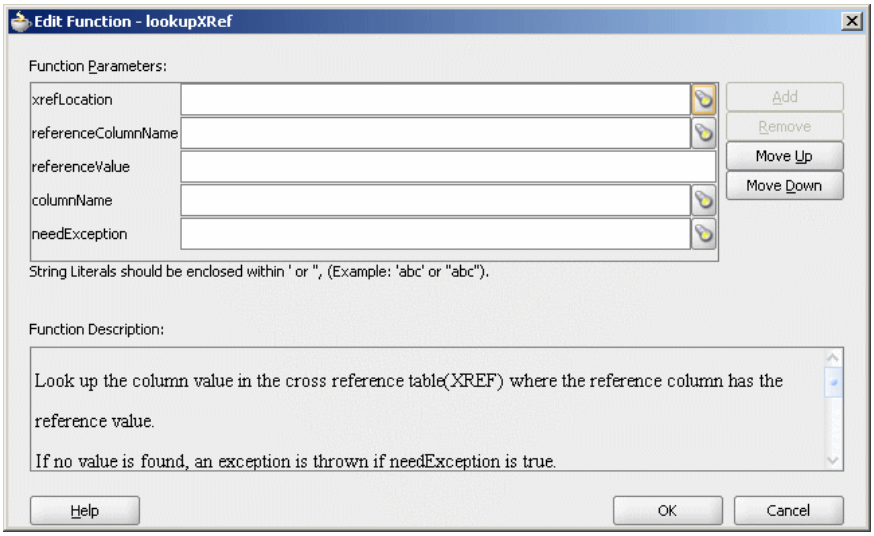
1. In the XSLT Mapper, expand the trees in the **Source** and **Target** panes.
2. Drag and drop the source element to the target element.

3. In the Components window, 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 43-10.

Figure 43-10 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.

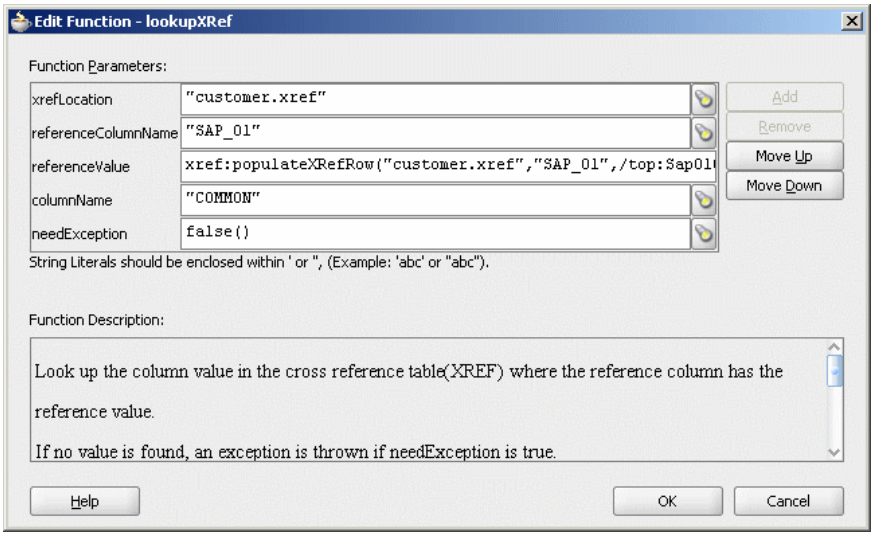
      Click **Browse** to the right of the **columnName** field to select a column name from the columns defined for the cross reference you previously selected.

   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 43-11.

Figure 43-11 Populated Edit Function – lookupXRef Dialog



Deleting a Cross Reference Table Value

You can use the `xref:markForDelete` function to delete a value in a cross reference table. The row, containing the column value passed to the function, is deleted from the `XREF_DATA` table and moved to the `XREF_DELETED_DATA` table. This function returns `true` if the deletion is successful. Otherwise, it returns `false`.

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 the following example:

```
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 that contains the value to be deleted.
- `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.

How to Delete a Cross Reference Table Value

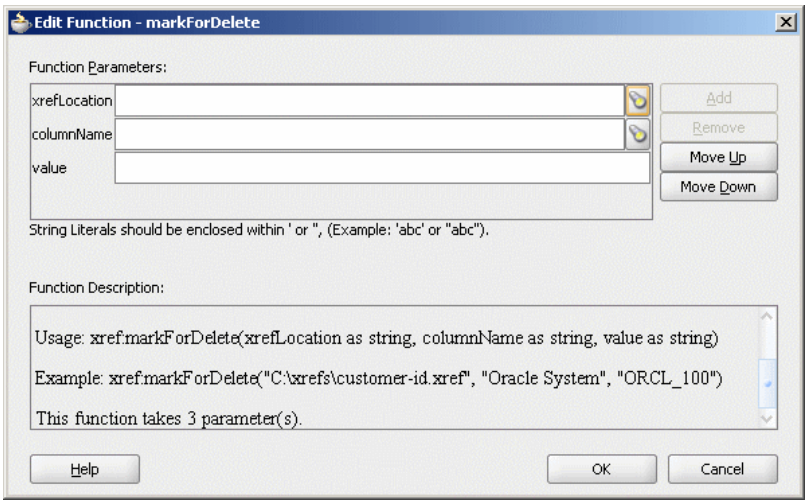
To delete a cross reference table value:

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 Components window, 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 43-12.

Figure 43-12 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.

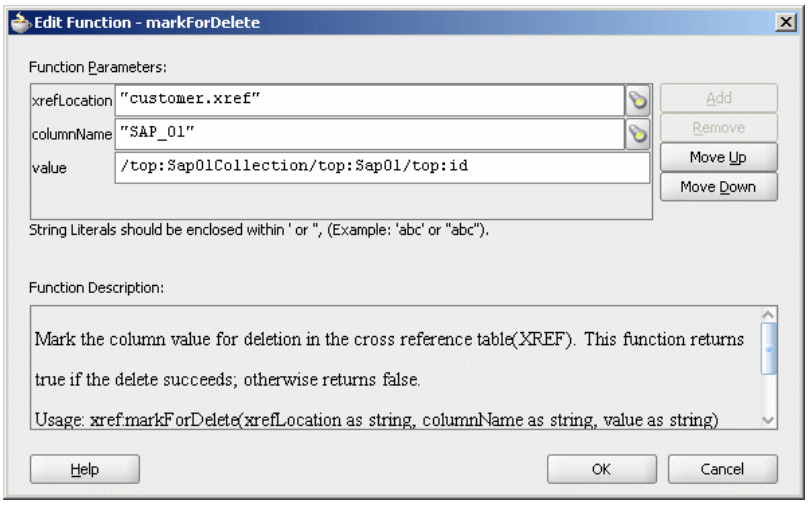
   b. In the **columnName** field, enter the name of cross reference table column.

      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 43-13.

Figure 43-13 Populated Edit Function – markForDelete Dialog

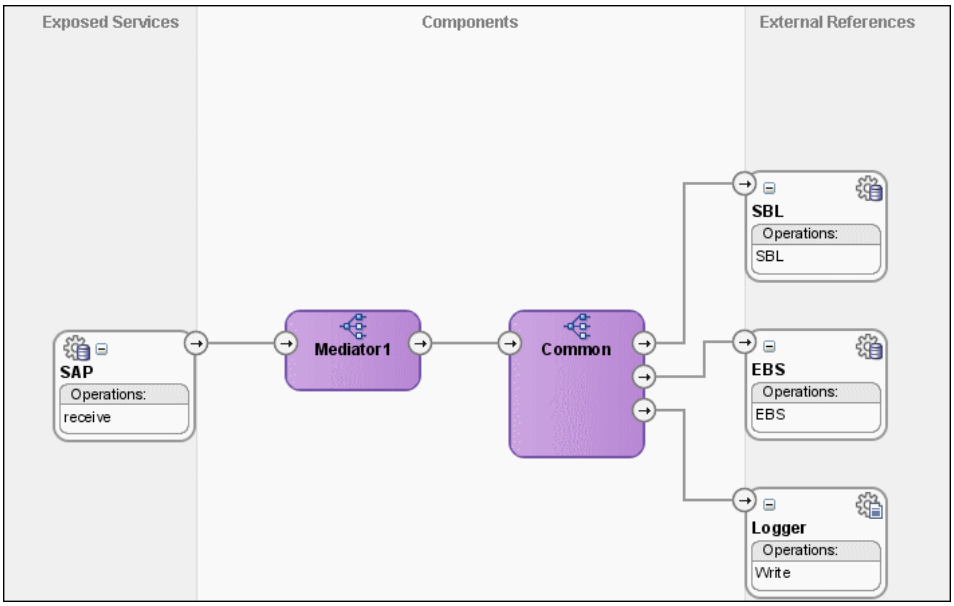


8. Click **OK**.

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 43-14 provides an overview of this use case.

Figure 43-14 XrefCustApp Use Case in



To download the sample files mentioned in this section, see the `Oracle SOA Suite samples page`.

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.

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`

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 Applications window 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**.

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 Applications window, 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**.

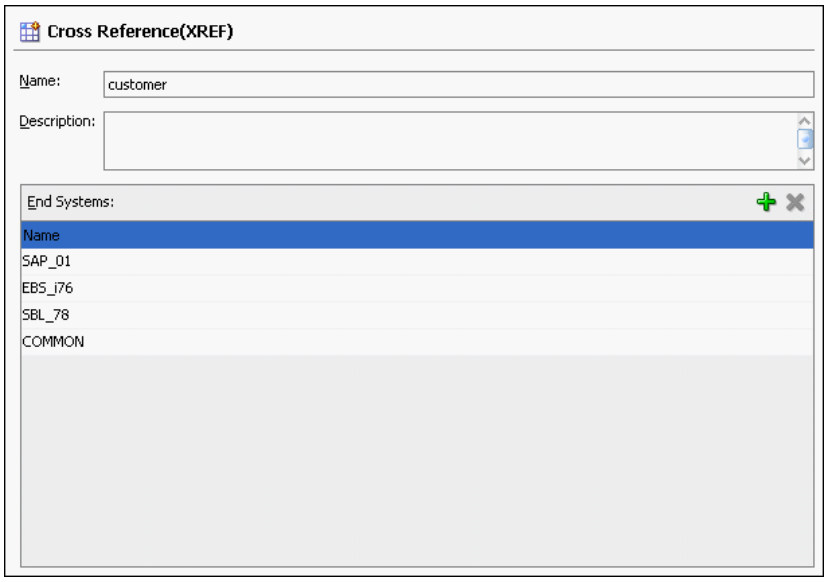
   The Create Cross Reference(XREF) File dialog is displayed.
4. In the **File Name** field, enter `customer.xref`.
5. In the **End System** fields, enter `SAP_01` and `EBS_i76`.
6. Click **OK**.

   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 43-15.

Figure 43-15 Customer Cross Reference



10. From the **File** menu, select **Save All** and close the Cross Reference Editor.

Task 4: How to Create a Database Adapter Service

To create a database adapter service:

1. In the Oracle JDeveloper Components window, select **SOA**.
2. Select **Database** 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**.

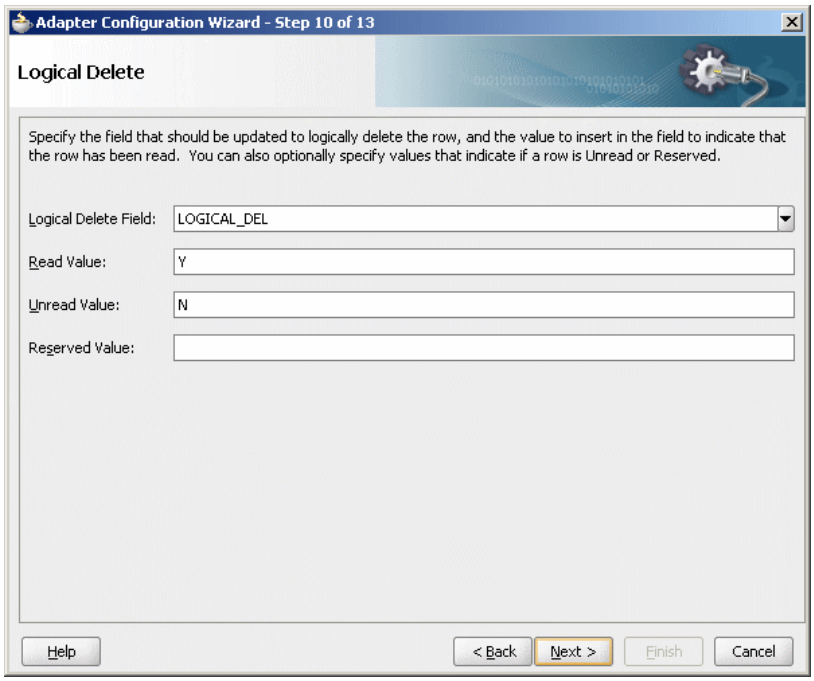
    The Attribute Filtering page is displayed.
18. Click **Next**.

    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 43-16 shows the Logical Delete page of the Adapter Configuration wizard.

Figure 43-16 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.

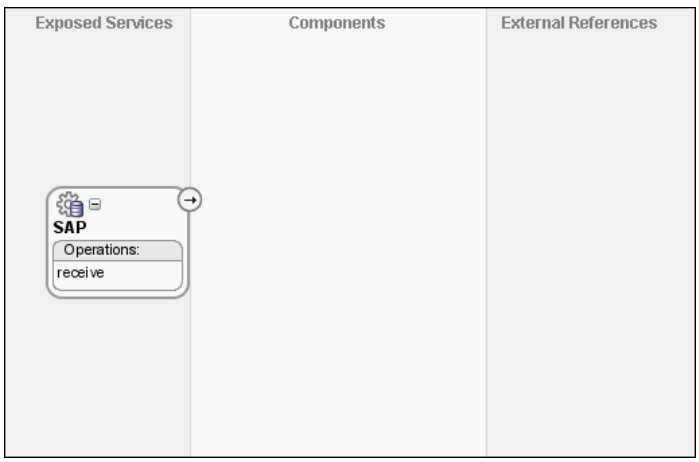
25. Click **Next**.

    The Finish page is displayed.

26. Click **Finish**.

    A database adapter service named **SAP** is created, as shown in Figure 43-17.

Figure 43-17 SAP Database Adapter Service in



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

Task 5: How to Create EBS and SBL External References

To create EBS and SBL external references:

1. In the Components window, 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**.

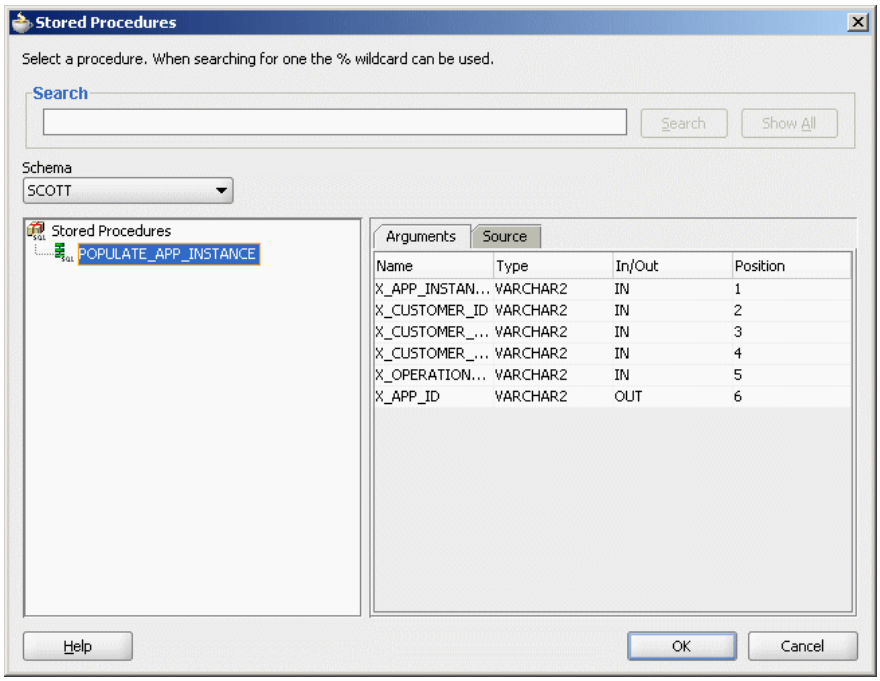
   The Service Connection page is displayed.
6. In the **Application Connection** field, select **DBConnection1**.
7. In the **JNDI Name** field, enter `eis/DB/DBConnection1`.
8. Click **Next**.

   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 43-18.

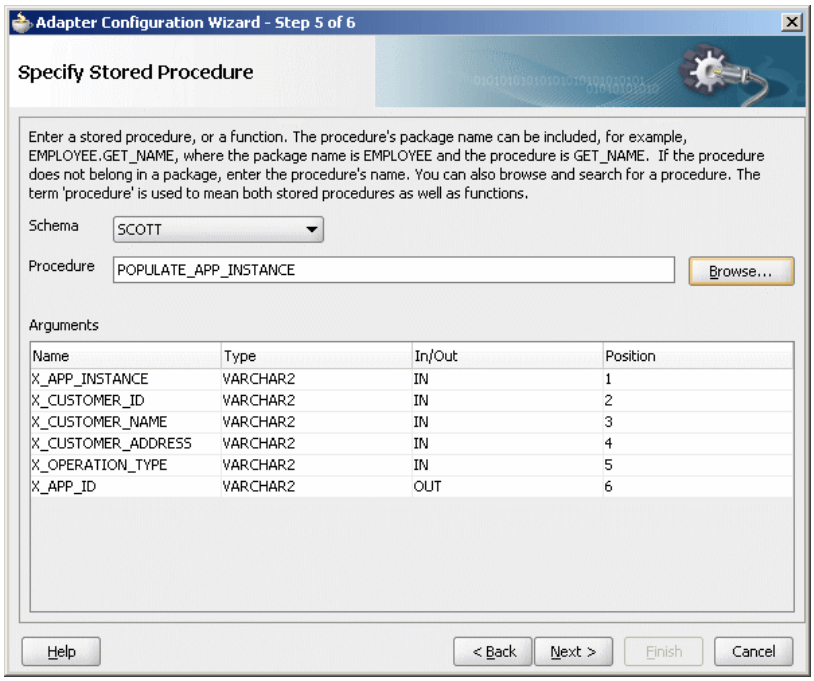
Figure 43-18 Stored Procedure Dialog



13. Click **OK**.

    The Specify Stored Procedure page appears, as shown in Figure 43-19.

Figure 43-19 Specify Stored Procedure Page of Adapter Configuration Wizard



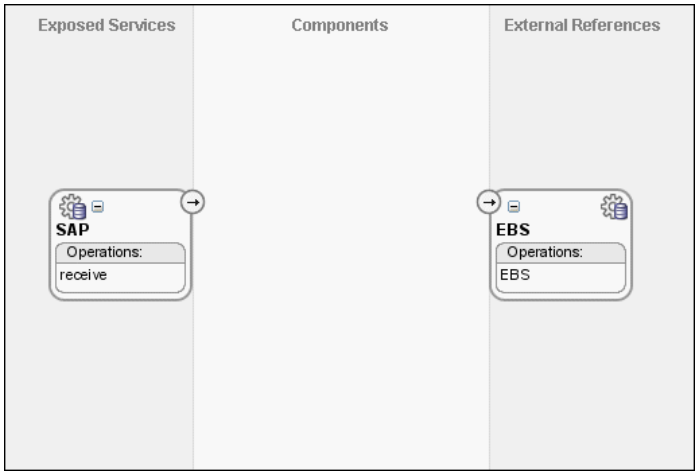
14. Click **Next**.

    The Finish page is displayed.

15. Click **Finish**.

Figure 43-20 shows the **EBS** reference in the .

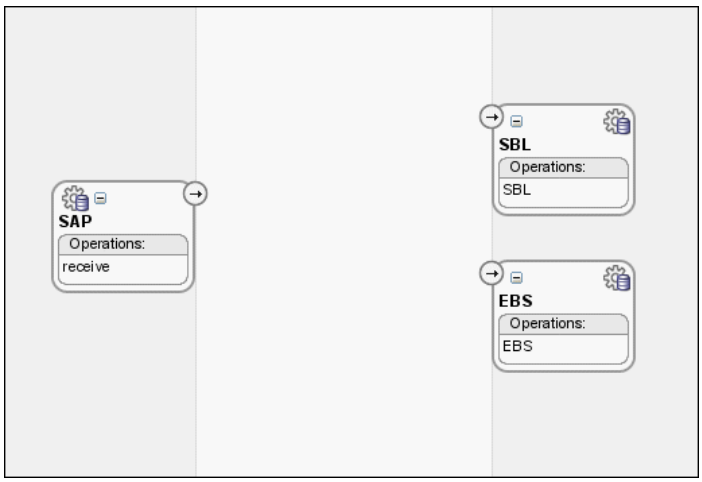
Figure 43-20 EBS Reference in



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 appears, as shown in Figure 43-21.

Figure 43-21 SBL Reference in



Task 6: How to Create the Logger File Adapter External Reference

To create the Logger file adapter external reference:

1. From the Components window, 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**.

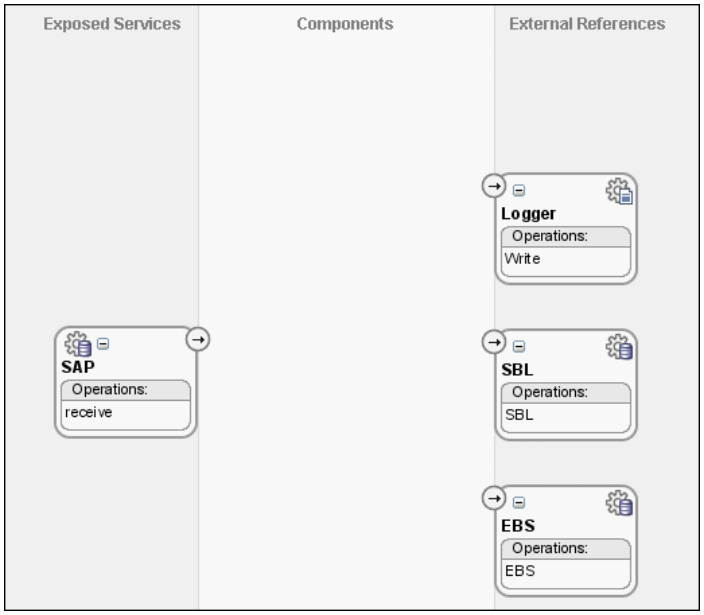
   The Messages page is displayed.
10. Click **Search**.

    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 43-22 shows the **Logger** reference in the .

Figure 43-22 Logger Reference in



15. From the **File** menu. select **Save All**.

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 Components window to the **Components** section of the SOA Composite Editor.

   The Create Mediator dialog is displayed.

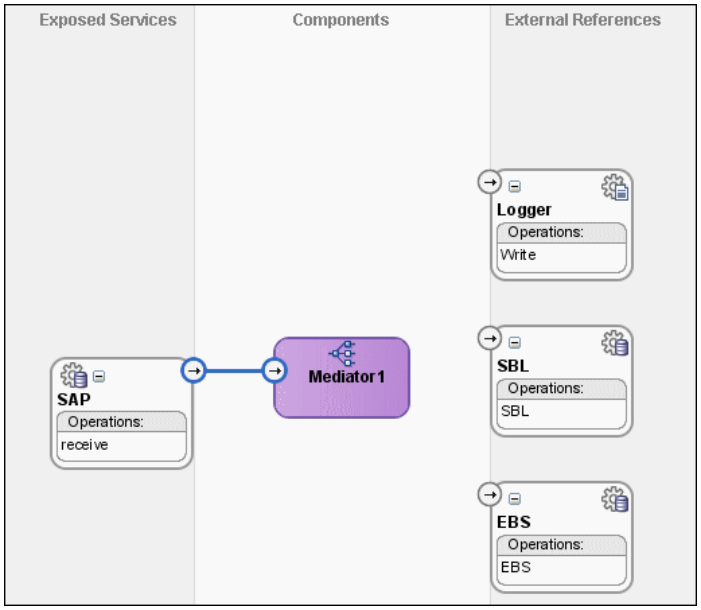
2. From the **Template** list, select **Define Interface Later**.

3. Click **OK**.

   An Oracle Mediator with name `Mediator1` is created.

4. Connect the **SAP** service to the **Mediator1**, as shown in Figure 43-23.

   Figure 43-23 SAP Service Connected to Mediator1



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

6. Drag and drop another **Mediator** icon from the Components window 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.

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.

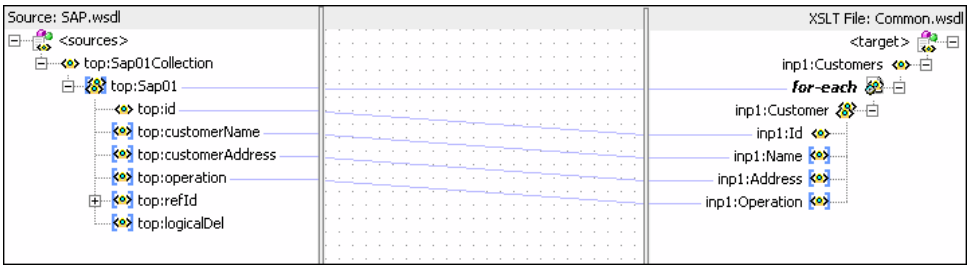
    The Auto Map Preferences dialog is displayed.

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 43-24.

Figure 43-24 SAP_TO_COMMON_INSERT.xsl Transformation



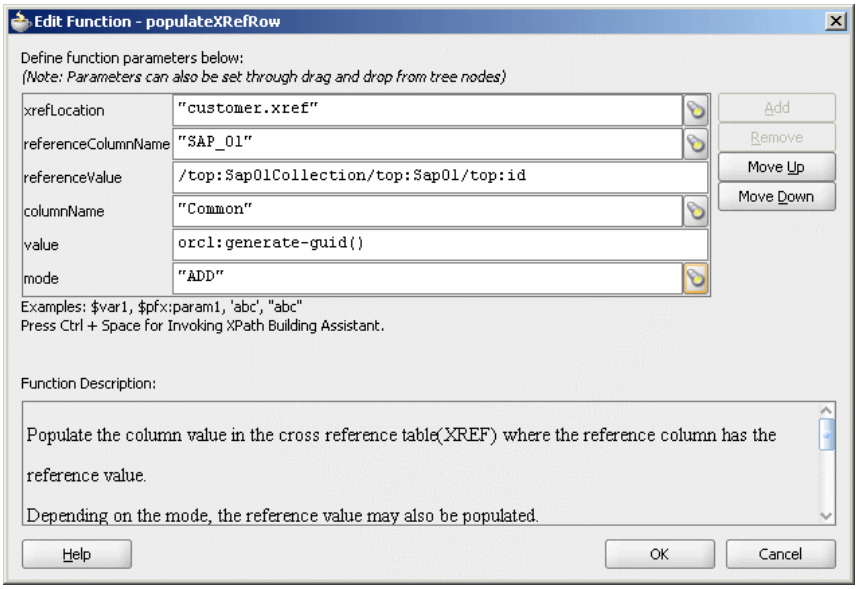
15. From the Components window, select **Advanced**.
16. Select **XREF Functions**.
17. Drag and drop the **populateXRefRow** function from the Components window to the line connecting the **top:id** and **inp1:id** elements.
18. Double-click the **populateXRefRow** icon.

    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 43-25 shows the populated Edit Function – populateXRefRow dialog.

Figure 43-25 Edit Function – populateXRefRow Dialog: XrefCustApp Use Case

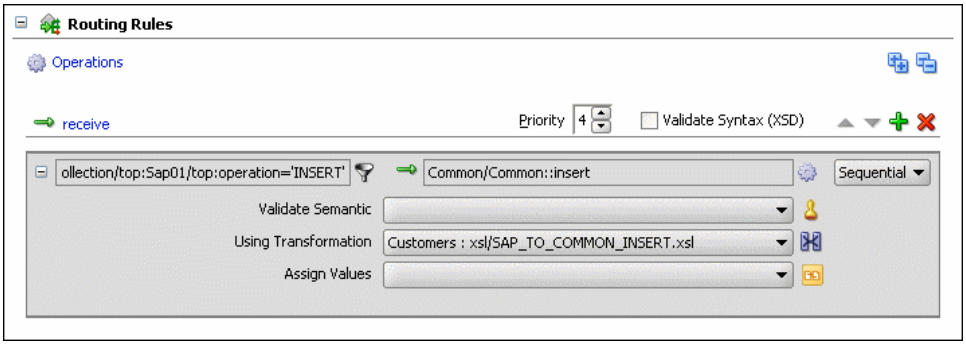


26. Click **OK**.

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 43-26.

Figure 43-26 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 Components window, select **Advanced**.

14. Select **XREF Functions**.

15. Drag and drop the **lookupXRef** function from the Components window 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.

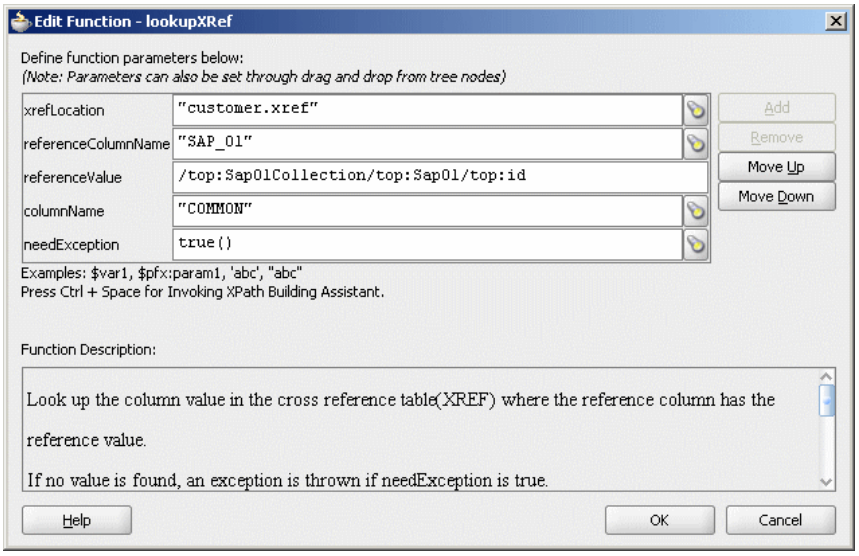
20. In the **referenceValue** column, enter `/top:Sap01Collection/top:Sap01/top:id`.

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 43-27 shows the populated Edit Function – looupXRef dialog.

Figure 43-27 Edit Function – lookupXRef Dialog: XrefCustApp Use Case

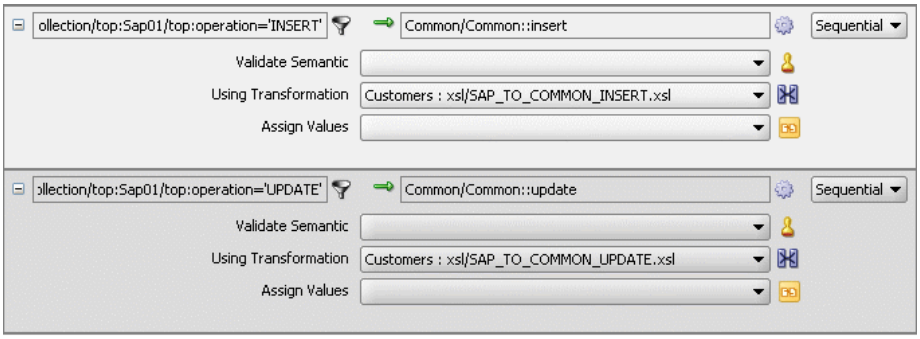


23. Click **OK**.

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 43-28.

Figure 43-28 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 Components window, select **Advanced**.

14. Select **XREF Functions**.

15. Drag and drop the **populateXRefRow** function from the Components window 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`.

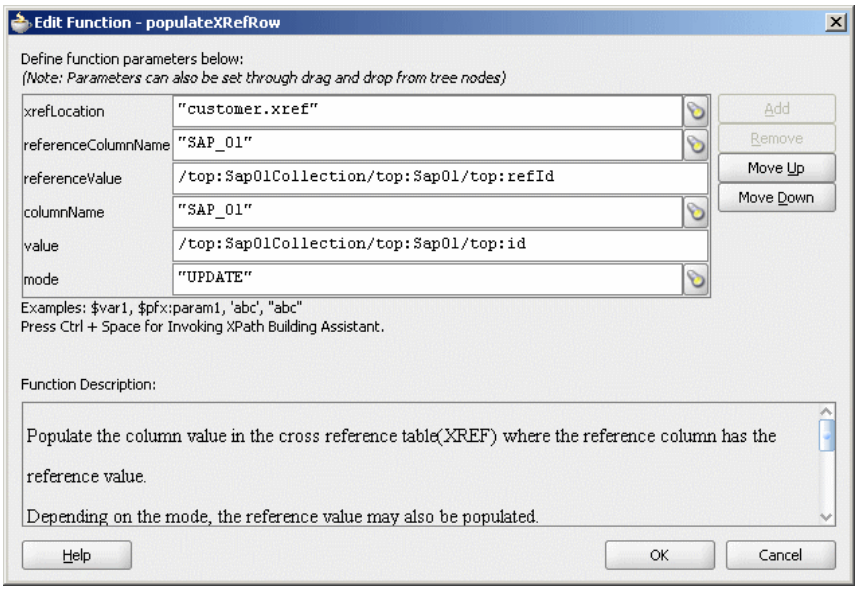
21. In the **columnName** field, enter `"SAP_01"` or click **Search** to select the column name.

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 43-29 shows a populated Edit Function – populateXRefRow dialog.

Figure 43-29 Edit Function – populateXRefRow Dialog: XrefCustApp Use Case



24. Drag and drop the **lookupXRef** function from the Components window 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:

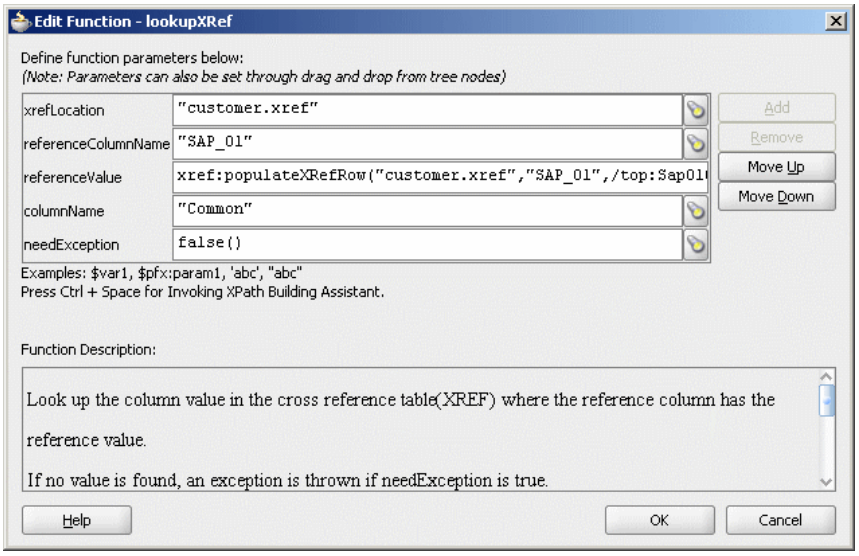
    ```
    xref:populateXRefRow("customer.xref","SAP_
    01",/top:Sap01Collection/top:Sap01/top:refId,"SAP_
    01",/top:Sap01Collection/top:Sap01/top:id,"UPDATE").
    ```

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 43-30 shows a populated Edit Function – lookupXRef dialog.

Figure 43-30 Edit Function – lookupXRef Dialog: XrefCustApp Use Case

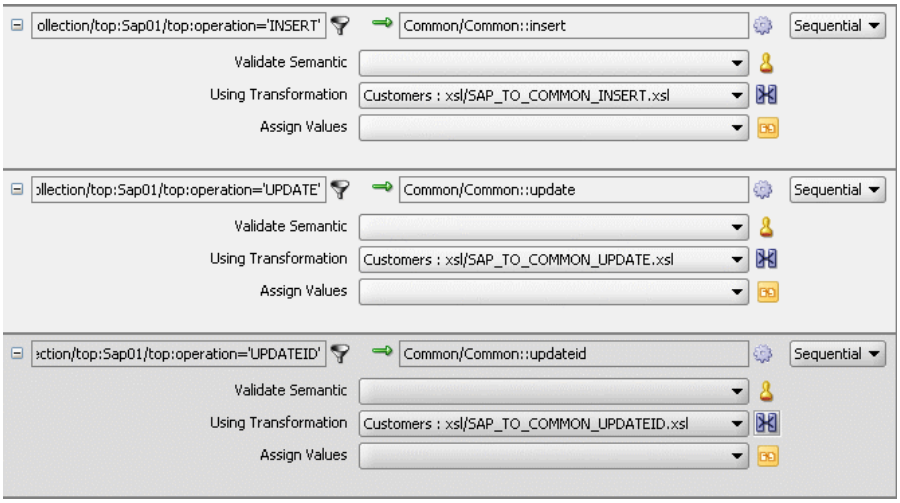


32. Click **OK**.

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 43-31.

Figure 43-31 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 Components window, select **Advanced**.

24. Select **XREF Functions**.

25. Drag and drop the **markForDelete** function from the Components window 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.

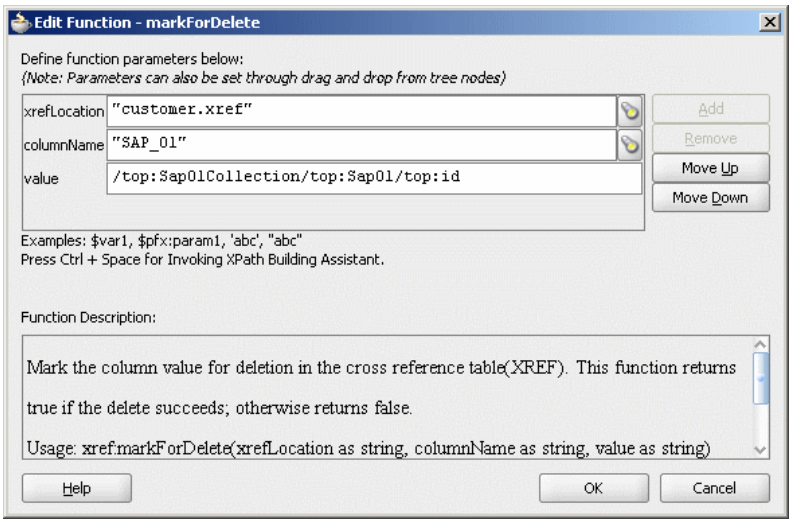
28. Select **customer.xref** and click **OK**.

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 43-32 shows a populated Edit Function – markForDelete dialog.

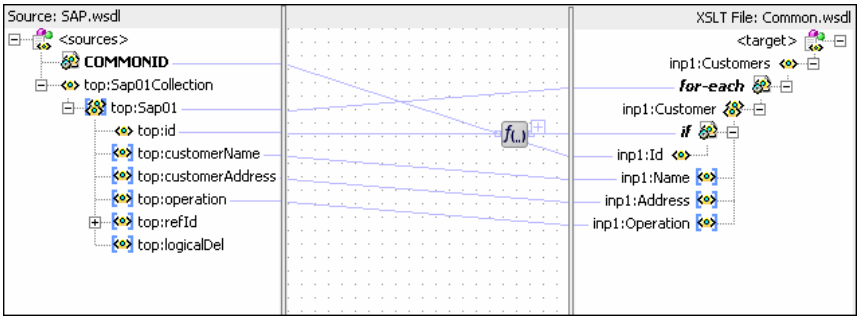
Figure 43-32 Edit Function – markForDelete Dialog: XrefCustApp Use Case



31. Click **OK**.

    The **SAP_TO_COMMON_DELETE.xsl** file appears, as shown in Figure 43-33.

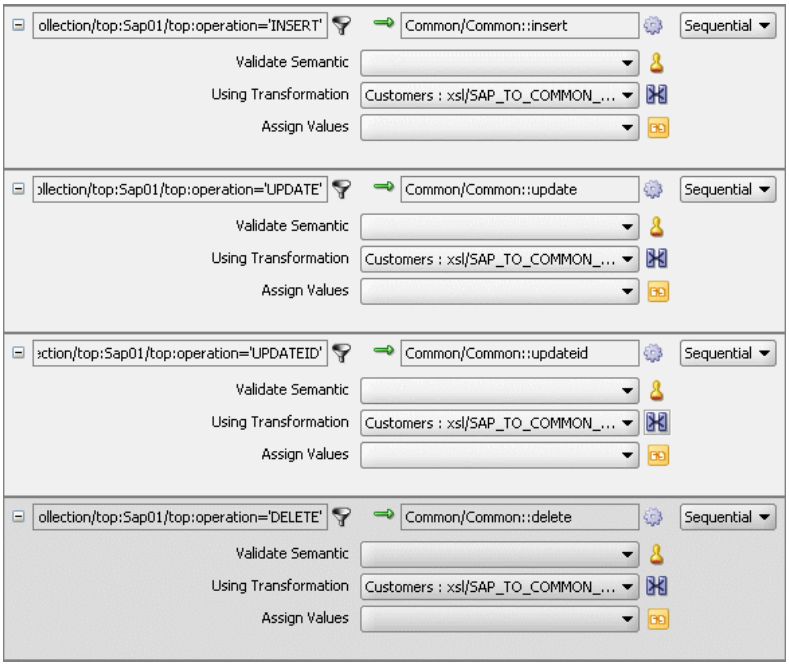
Figure 43-33 SAP_TO_COMMON_DELETE.xsl



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 43-34.

Figure 43-34 Insert, Update, UpdateID, and Delete Operations



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.

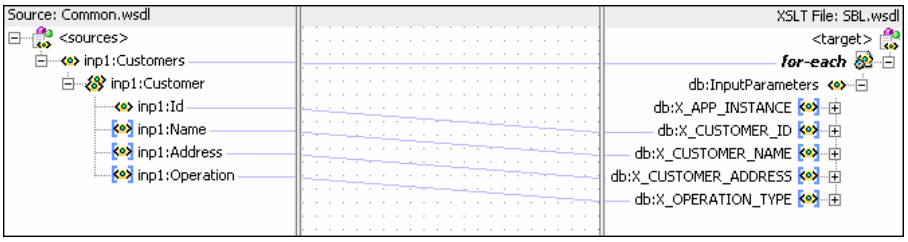
9. Drag and drop the **inp1:Customers** source element to the **db:InputParameters** target element.

   The Auto Map Preferences dialog is displayed.

10. Click **OK**.

    The transformation is created, as shown in Figure 43-35.

Figure 43-35 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 Components window 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:

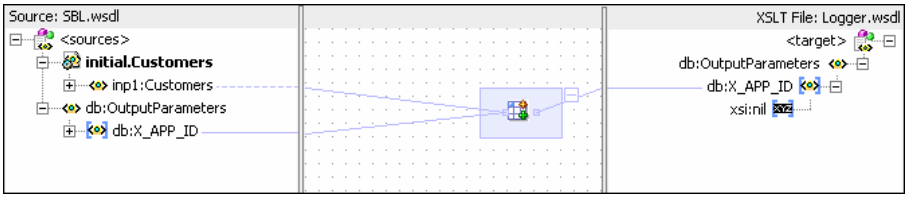
    - **xrefLocation**: `"customer.xref"`

- **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 43-36.

Figure 43-36 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**.

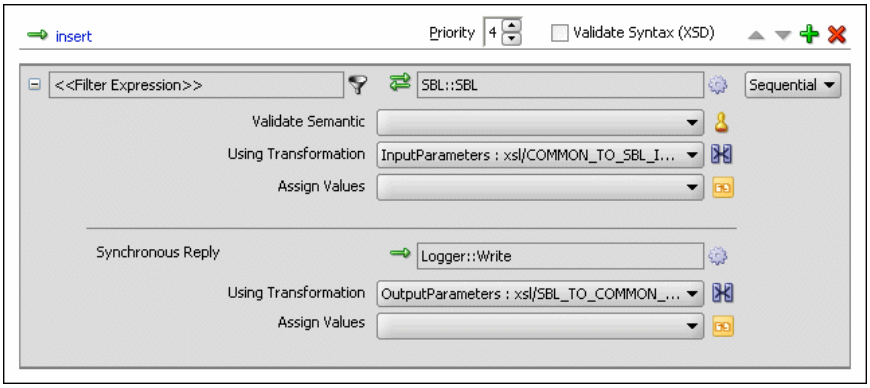
    The Assign Value dialog is displayed.
28. In the **From** section, select **Expression**.
29. Click the **Invoke Expression Builder** icon.

    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 43-37.

Figure 43-37 Insert Operation with SBL Target Service

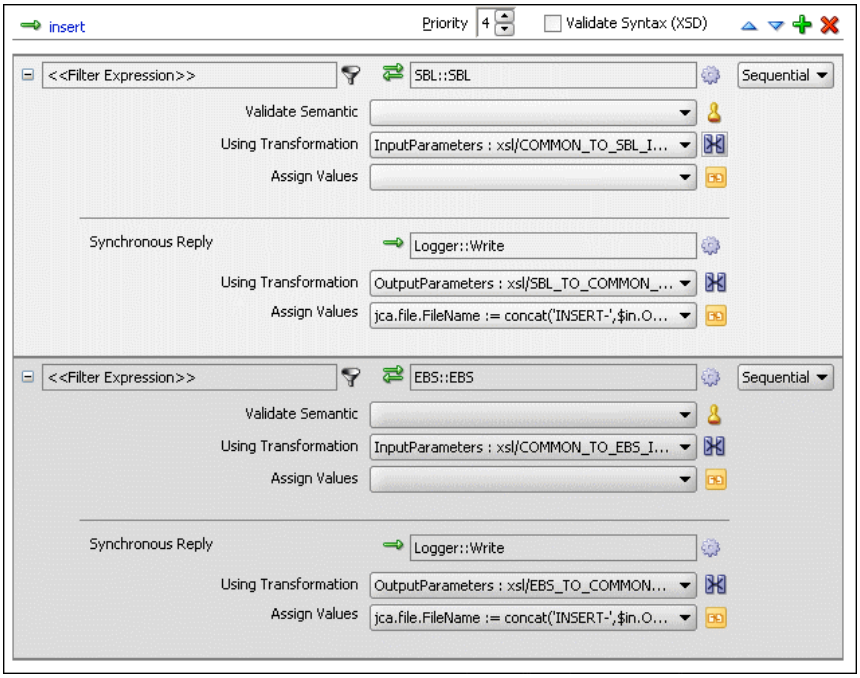


34. From the **File** menu, select **Save All**.

35. Repeat Step 2 through Step 34 to specify another target service named **EBS** and its routing rules.

    Figure 43-38 shows the **insert** operation section with **SBL** and **EBS** target services.

Figure 43-38 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.

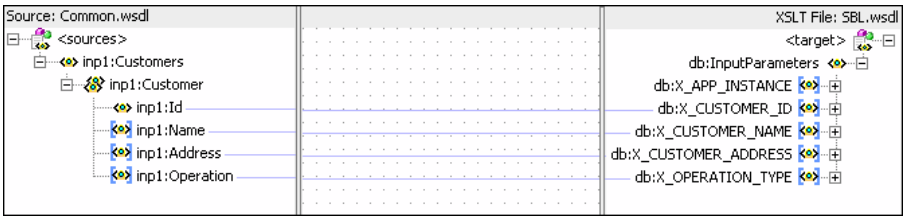
   The Request Transformation map dialog is displayed.
6. Select **Create New Mapper File** and enter `COMMON_TO_SBL_DELETE.xsl`.
7. Click **OK**.

   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 43-39.

Figure 43-39 COMMON_TO_SBL_DELETE.xsl Transformation



10. Drag and drop the **lookupXRef** function from the Components window 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 Components window 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**.

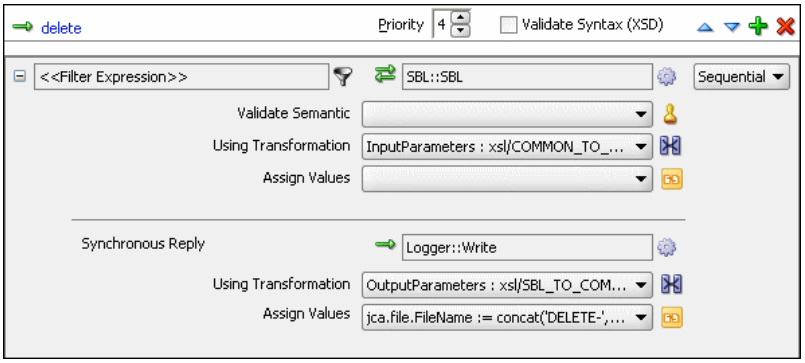
    The Assign Value dialog is displayed.
30. In the **From** section, select **Expression**.
31. Click the **Invoke Expression Builder** icon.

    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 43-40.

Figure 43-40 Delete Operation with SBL Target Service

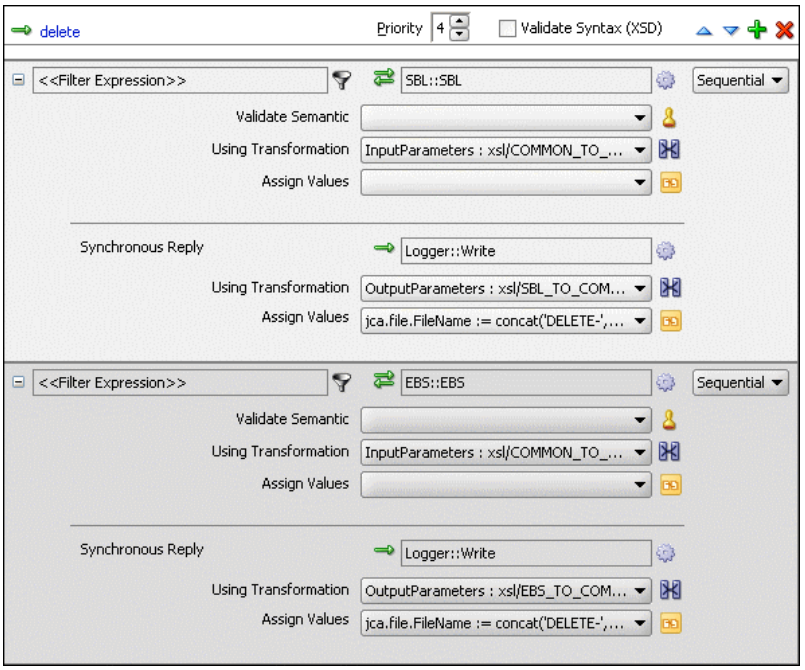


36. From the **File** menu, select **Save All**.

37. Repeat Step 1 through Step 36 to specify another target service named **EBS** and specify the routing rules.

    Figure 43-41 shows the **delete** operation section with **SBL** and **EBS** target services.

Figure 43-41 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 43-39.
10. Drag and drop the **lookupXRef** function from the Components window 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**.

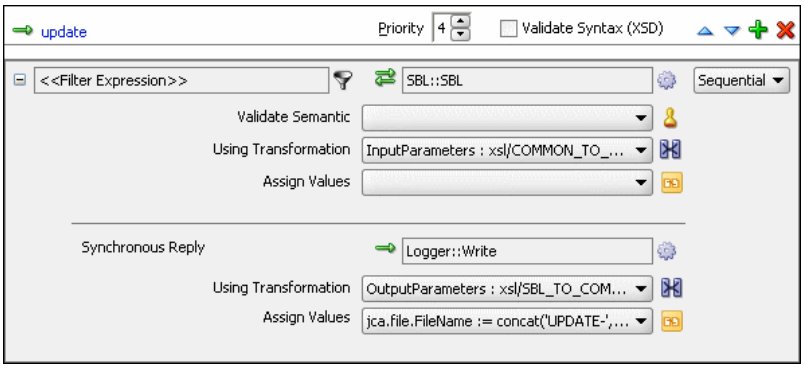
    The Assign Value dialog is displayed.
26. In the **From** section, select **Expression**.
27. Click the **Invoke Expression Builder** icon.

    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 43-42.

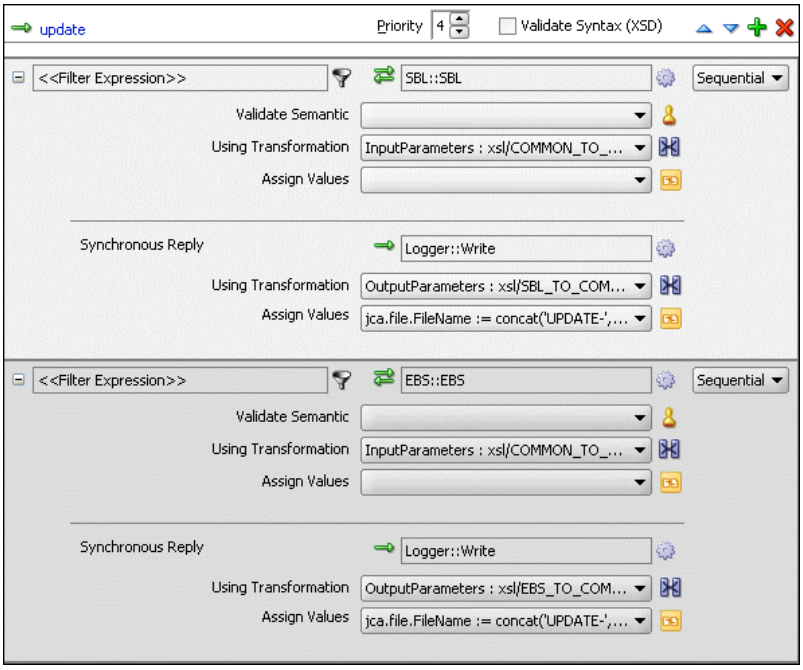
Figure 43-42 Update Operation with SBL Target Service



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 43-43 shows the **update** operation section with **SBL** and **EBS** target services.

Figure 43-43 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 43-39.

10. Drag and drop the **lookupXRef** function from the Components window 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 Components window 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.

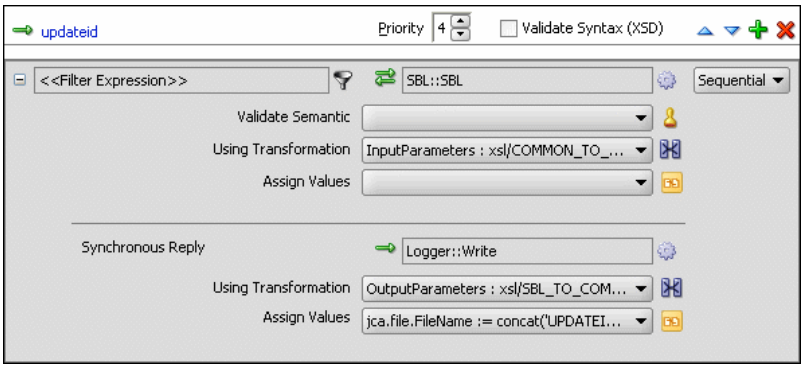
    The Expression Builder dialog is displayed.
32. In the **Expression** field, enter the following expression and click **OK**.

```
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 43-44.

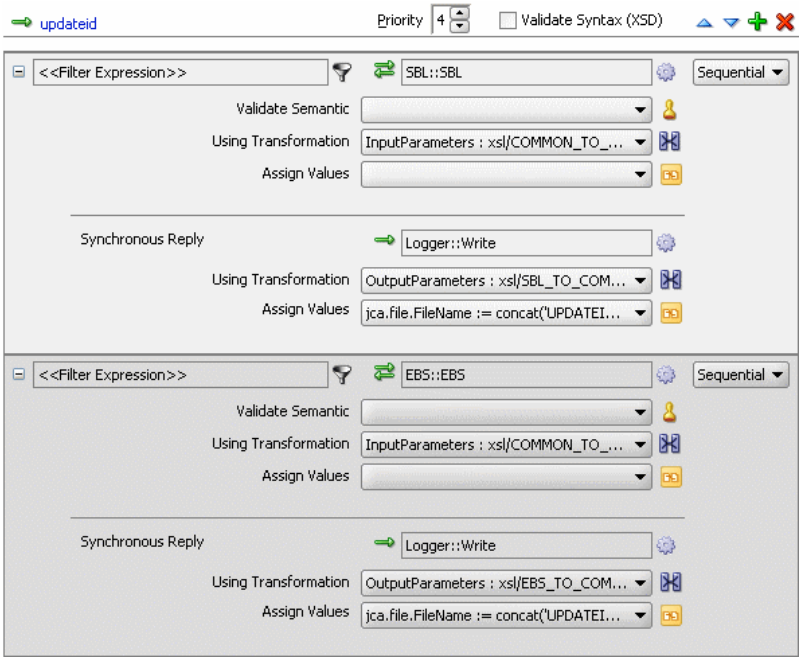
Figure 43-44 Updateid Operation with SBL Target Service



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 43-45 shows the **updateid** operation section with the **SBL** and **EBS** target services.

Figure 43-45 Updateid Operation with SBL and EBS Target Service



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 Creating an Application Server Connection.

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 How to Deploy a Single SOA Composite in .

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:

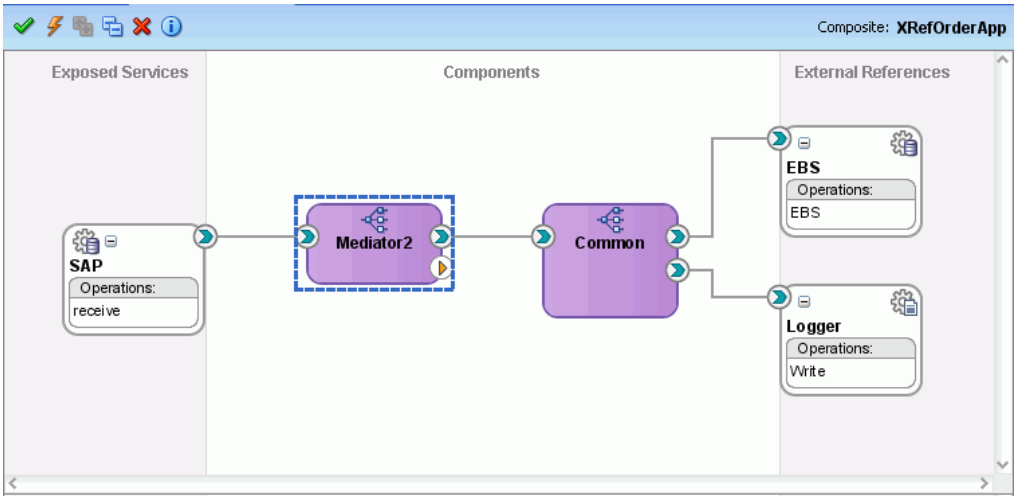
`http://hostname:port_number/em`

where *`hostname`* is the host on which you installed the Oracle SOA Suite infrastructure and *`port_number`* is the port running the service.

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 43-46 provides an overview of this use case.

Figure 43-46 XrefOrderApp Use Case in



To download the sample files mentioned in this section, see the `Oracle SOA Suite samples page`.

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.

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`

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 Applications window 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**.

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 Applications window, 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**.

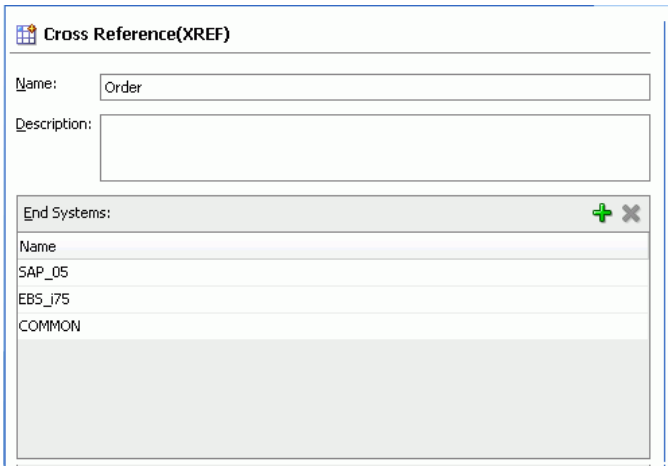
   The Create Cross Reference(XREF) File dialog is displayed.
4. In the **File Name** field, enter `order.xref`.
5. In the **End System** fields, enter `SAP_05` and `EBS_i75`.
6. Click **OK**.

   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 43-47.

Figure 43-47 Customer Cross Reference



9. From the **File** menu, select **Save All** and close the Cross Reference Editor.

Task 4: How to Create a Database Adapter Service

To create a database adapter service:

1. In the Components window, 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 43-16 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.

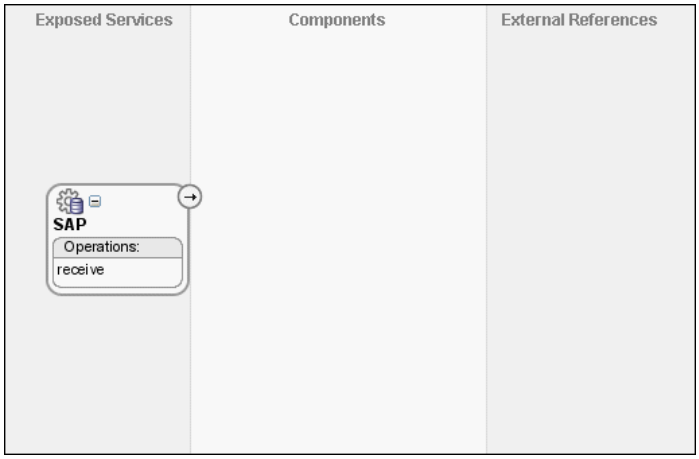
26. Click **Next**.

    The Finish page is displayed.

27. Click **Finish**.

    A database adapter service named **SAP** is created, as shown in Figure 43-48.

Figure 43-48 SAP Database Adapter Service in



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

Task 5: How to Create an EBS External Reference

To create an EBS external reference:

1. In the Components window, 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**.

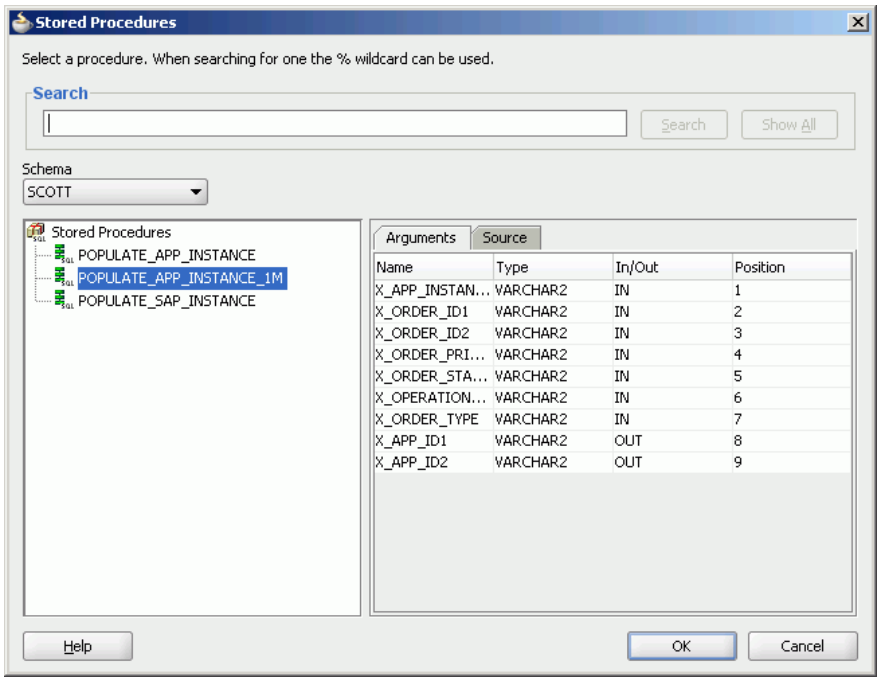
   The Service Connection page is displayed.
6. In the **Connection** field, select **DBConnection1**.
7. In the **JNDI Name** field, enter `eis/DB/DBConnection1`.
8. Click **Next**.

   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 43-49.

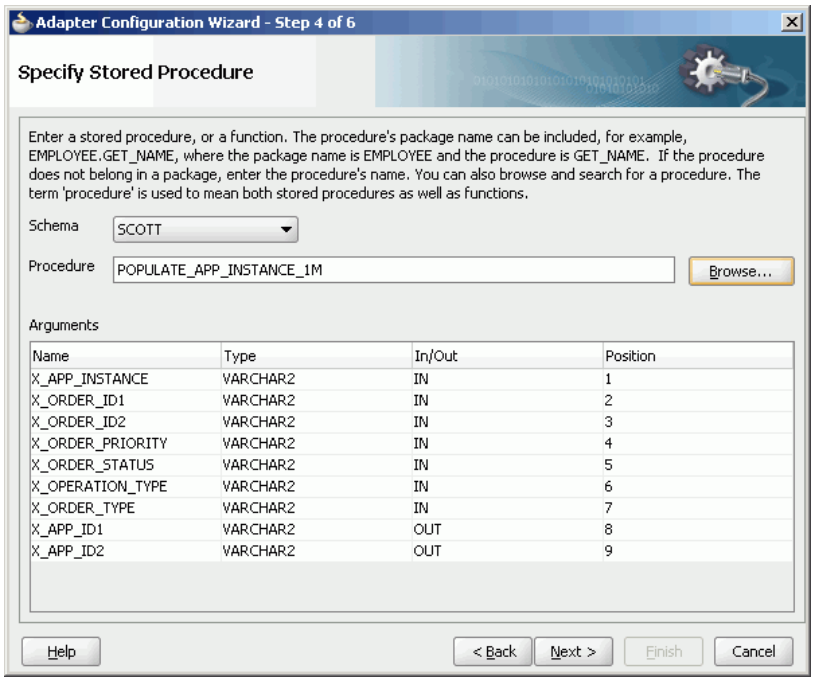
Figure 43-49 Stored Procedure Dialog



13. Click **OK**.

    The Specify Stored Procedure page appears, as shown in Figure 43-50.

Figure 43-50 Specify Stored Procedure Page of Adapter Configuration Wizard



14. Click **Next**.

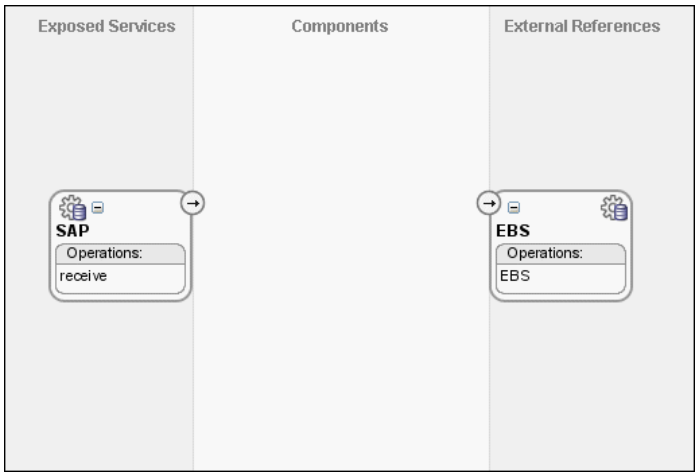
    The Advanced Options page is displayed.

15. Click **Next**.The Finish page is displayed.

16. Click **Finish**.

    Figure 43-51 shows the **EBS** reference in the .

Figure 43-51 EBS Reference in



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

Task 6: How to Create a Logger File Adapter External Reference

To create a Logger file adapter external reference:

1. From the Components window, 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**.

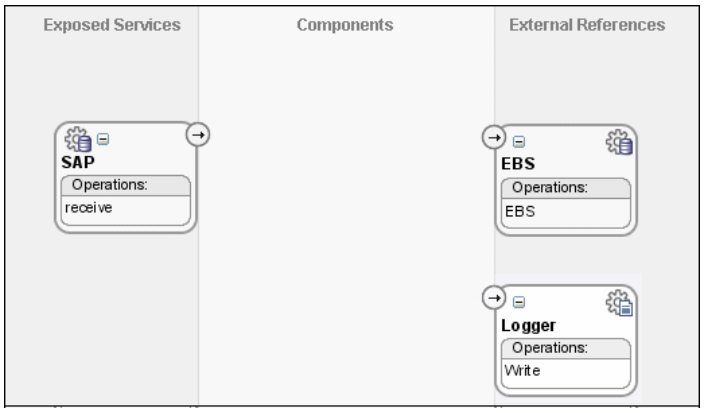
14. Click **Next.**

    The Finish page is displayed.

15. Click **Finish**.

    Figure 43-52 shows the **Logger** reference in the .

    Figure 43-52 Logger Reference in



16. From the **File** menu. select **Save All**.

Task 7: How to Create an Oracle Mediator Service Component

To create an service component:

1. Drag and drop a **Mediator** icon from the Components window to the **Components** swimlane.

   The Create Mediator dialog is displayed.

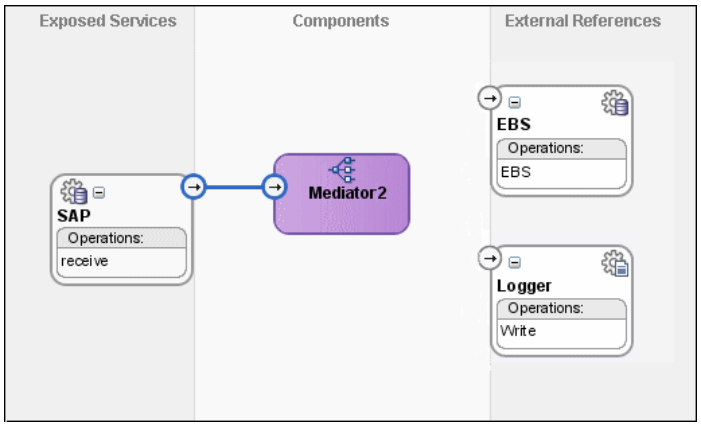
2. From the **Template** list, select **Define Interface Later**.

3. Click **OK**.

   An Oracle Mediator with name **Mediator2** is created.

4. Connect the **SAP** service to **Mediator2**, as shown in Figure 43-53.

Figure 43-53 SAP Service Connected to Mediator2



5. From the **File** menu. select **Save All**.
6. Drag and drop a Mediator icon from the Components window 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.

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.

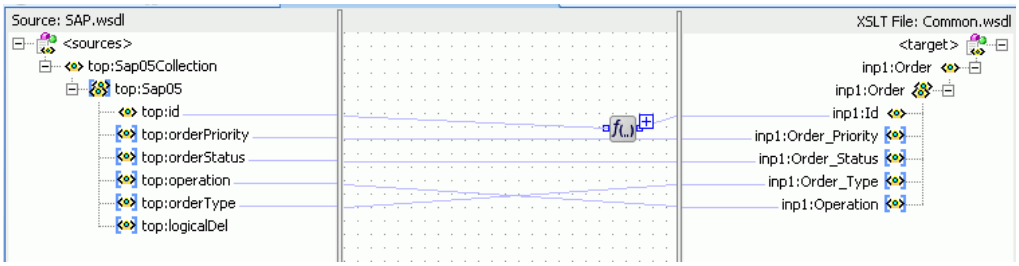
   The Request Transformation map dialog is displayed.
10. Select **Create New Mapper File** and enter `SAP_TO_COMMON_INSERT.xsl`.
11. Click **OK**.

    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 43-54.

Figure 43-54 SAP_TO_COMMON_INSERT.xsl Transformation



15. From the Components window, select **Advanced**.
16. Select **XREF Functions**.
17. Drag and drop the **populateXRefRow1M** function from the Components window 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`.

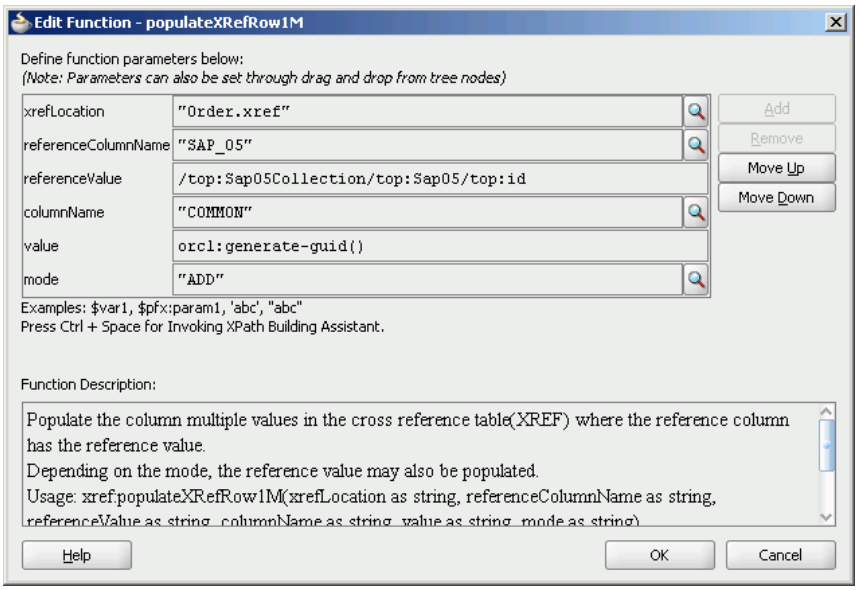
23. In the **columnName** field, enter `"Common"` or click **Search** to select the column name.

24. In the **value** field, enter `orcl:generate-guid()`.

25. In the **mode** field, enter `"Add"` or click **Search** to select this mode.

    Figure 43-55 shows the populated Edit Function – populateXRefRow1M dialog.

Figure 43-55 Edit Function – populateXRefRow1M Dialog: XrefOrderApp Use Case

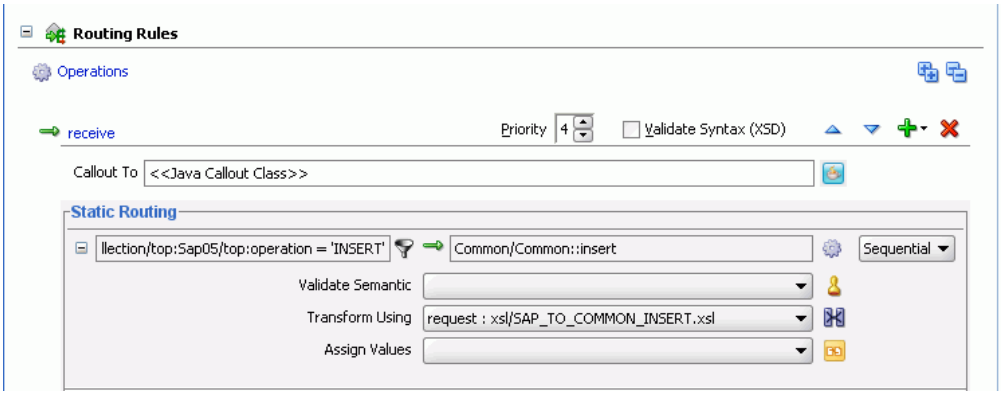


26. Click **OK**.

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 43-56.

Figure 43-56 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 Components window, select **Advanced**.

14. Select **XREF Functions**.

15. Drag and drop the **lookupXRef** function from the Components window 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.

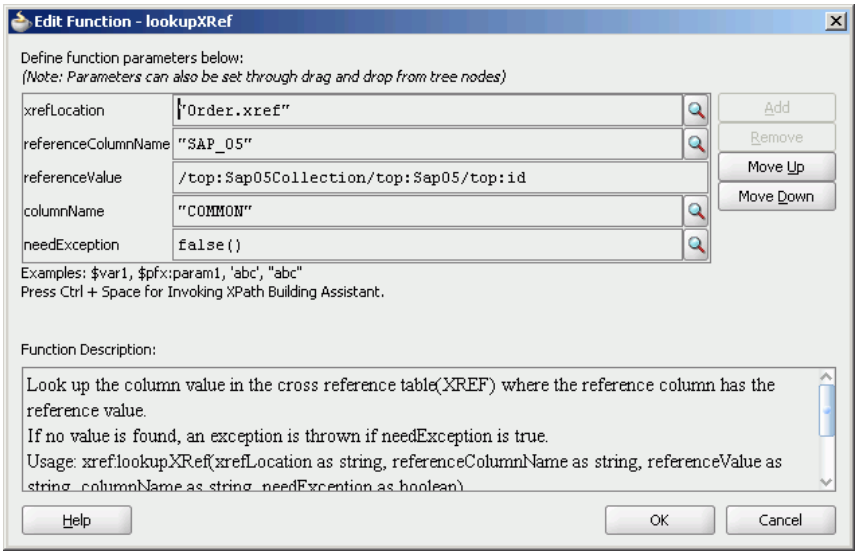
20. In the **referenceValue** column, enter `/top:Sap05Collection/top:Sap05/top:id`.

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 43-57 shows the populated Edit Function – looupXRef dialog.

Figure 43-57 Edit Function – looupXRef Dialog: XRefOrderApp Use Case

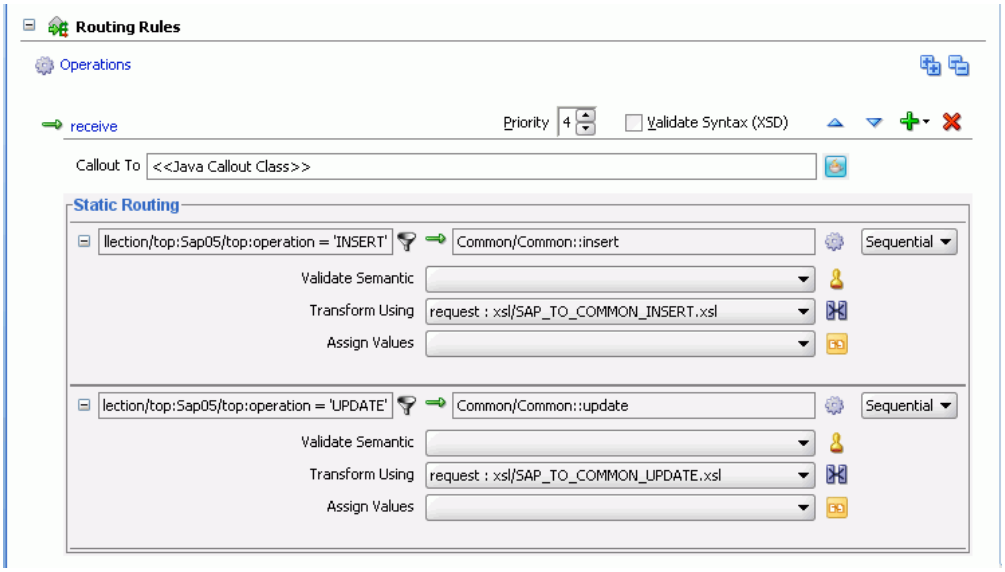


23. Click **OK**.

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 43-58.

Figure 43-58 Insert Operation and Update Operation



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.

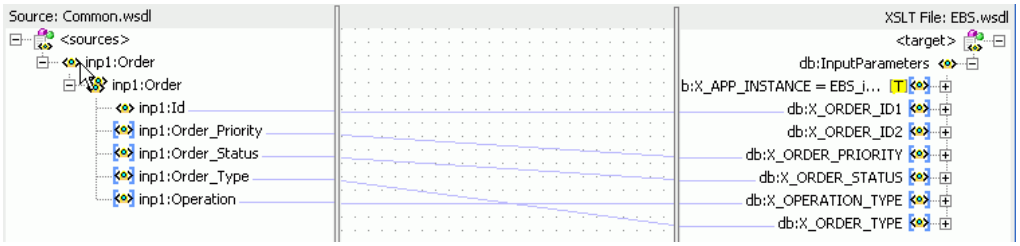
   The Auto Map Preferences dialog is displayed.

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 43-59.

Figure 43-59 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.

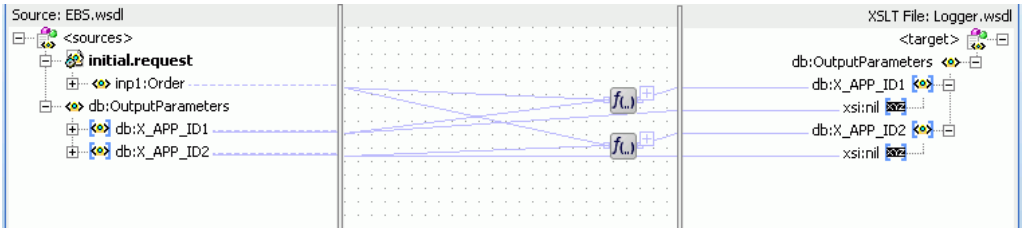
    The Reply Transformation map dialog is displayed.
17. Select **Create New Mapper File** and enter `EBS_TO_COMMON_INSERT.xsl`.
18. Select **Include Request in the Reply Payload**.
19. Click **OK**.

    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 Components window 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 43-60.

Figure 43-60 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.

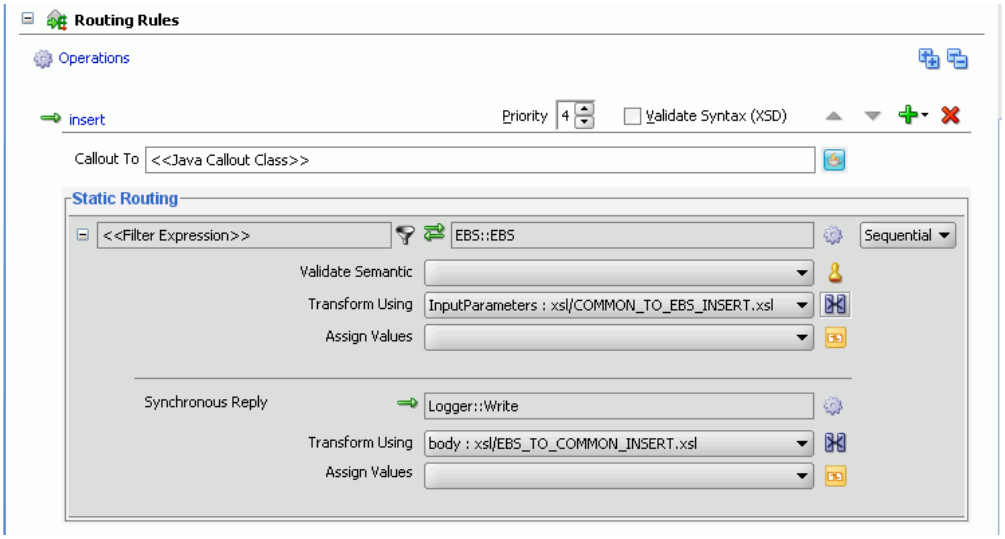
28. In the **From** section, select **Expression**.
29. Click the **Invoke Expression Builder** icon.

    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 43-61.

Figure 43-61 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 43-39.

10. Drag and drop the **lookupXRef** function from the Components window 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')
    ```

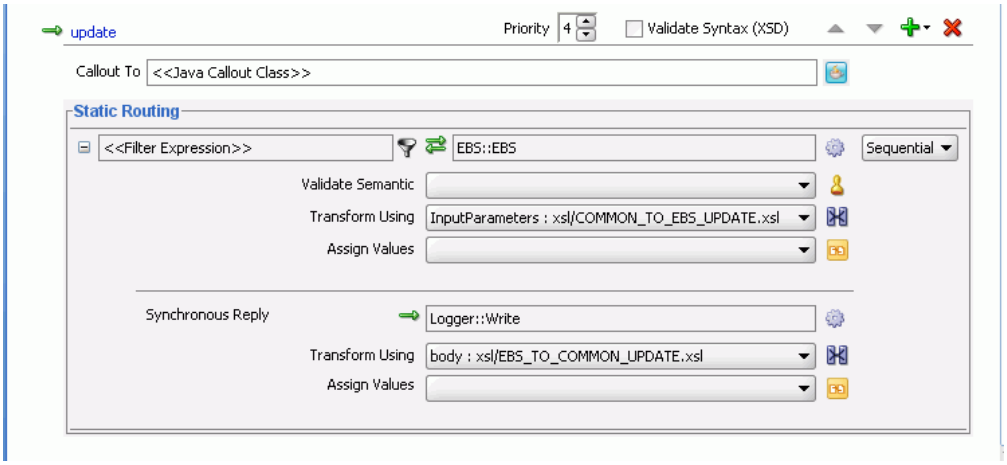
29. In the **To** section, select **Property**.

30. Select the **jca.file.FileName** property and click **OK**.

31. Click **OK**.

    The **update** operation section appears, as shown in Figure 43-62.

Figure 43-62 Update Operation with EBS Target Service



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

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 olink:TKADP212Creating an Application Server Connection.

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 How to Deploy a Single SOA Composite in .

44

Working with Domain Value Maps

This chapter describes how to create and use domain value maps to map the terms used by different domains to describe the same entity, allowing you to map values used by one domain for specific fields to the values used by other domains for the same fields. This chapter also describes the XPath functions used for domain value lookups.

This chapter includes the following sections:

Introduction to Domain Value Maps

When information is transmitted between different domains, each domain might use different terminology or processing codes to describe the same entity. For example, one domain might use complete city names in its messages (`Boston`), while another domain uses a code to indicate the city (`BO`). Rather than requiring each domain to standardize their data to one set of terminology, you can use domain value maps to map the terms used in one domain to the terms used in other domains. Domain value maps operate on the actual data values in the messages that are transmitted through an application at runtime.

While each domain value map typically defines the mapping for only one field or category, a single SOA composite can require mappings for multiple categories. Thus, one SOA composite might contain several domain value maps. For example, you might have one domain value map that defines city name mapping, one that defines state name mapping, and one that defines country name mapping.

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`

Table 44-1 (Cont.) Point-to-Point Mapping

CityCode	CityName
BO	Boston
NP	Northport
KN_USA	KensingtonUSA
KN_CAN	KensingtonCanada

Domain value map values are static. You specify the domain value map values at design time using Oracle JDeveloper, and then at runtime the application performs a lookup for the values in the domain value maps. For information about editing domain value maps at runtime with Oracle SOA Composer, see Using 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 Working with Cross References .

Domain Value Map Features

Oracle SOA Suite domain value maps let you further refine the performance and results of the domain value map lookups that are performed at runtime. For example, you can specify qualifying information that provides additional information to assist with mapping. Domain value maps also support one-to-many mappings.

Qualifier Domains

Qualifier domains contain information solely to clarify the mapping. A mapping might be ambiguous unless this additional information is defined. For example, a domain value map that defines a city name mapping could 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 indicate which mapping to use. An example of this is 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

A domain value map can contain multiple qualifier domains. For example, as shown in Table 44-3, the mappings are also 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.

Qualifier Hierarchies

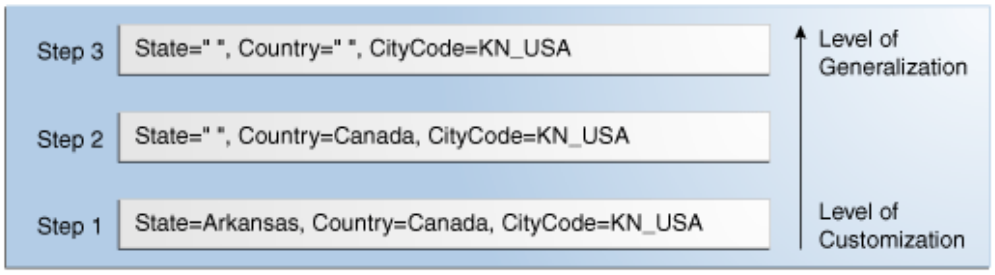
When there are multiple qualifier domains, you can specify a qualifier order to indicate how they are used during runtime lookups. The order of a qualifier varies from highest to lowest depending on the role of the qualifier in defining a more exact match. In Table 44-3, the state qualifier is probably given a higher order than the country qualifier because a matching state indicates a more precise 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 empty quotes (""). It proceeds until a match is found, or until a the lookup is exhausted and no match is found. Figure 44-1 describes the steps of a hierarchical lookup performed for the following lookup (based on the values in Table 44-3):

```
State=Arkansas, Country=Canada, CityCode=KN_USA
```

In this example, the `State` qualifier has a qualifier order of `1` and the `Country` qualifier has a qualifier order 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



If 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`. If

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`

One-to-Many Mappings

One value can be mapped 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 multiple 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`

Creating Domain Value Maps

You can create one or more domain value maps in a SOA composite application in Oracle JDeveloper, and then use the maps to look up the mapped values at runtime. Creating a domain value map creates a file with a `.dvm` extension in the application file structure.

How to Create Domain Value Maps

Create and configure domain value maps using the Create Domain Value Map(DVM) File dialog in Oracle JDeveloper. This dialog lets you define two domains, each with one value. Upon completion, the Domain Value Map Editor appears so you can define additional domains and corresponding values.

To create a domain value map:

1. In the Applications window, right-click the project in which you want to create a domain value map and select **New**.

   The New Gallery dialog appears.

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 appears.

4. In the **File Name** field, enter a unique and descriptive name for the domain value map file. The file name must have an extension of **.dvm**.
5. In the **Description** field, enter a description for the domain value map. This field is optional.
6. In the **Domain Name** field, enter a name for each domain. These names are the column names for the domain value map, and each represents a fields in a different domain.

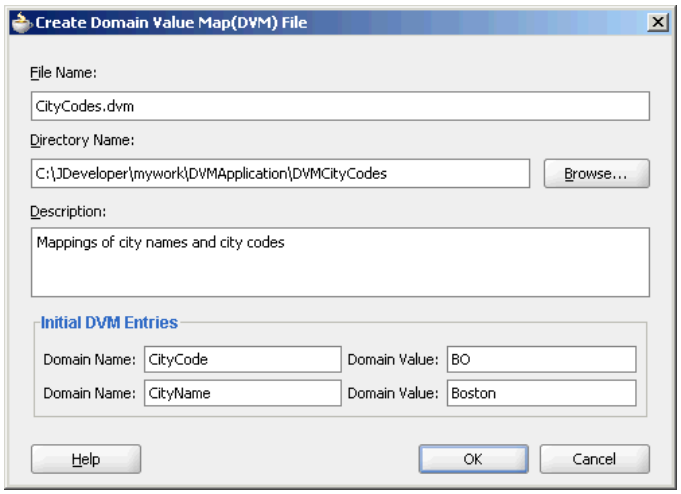
Note:

Domain names must be of the type NCName (non-colonized name), which is a valid XML element name with no colons. Each domain name must be unique in a domain value map. You can add more domains later.

---

7. In the **Domain Value** field, enter a value corresponding to each domain. For example, enter `BO` for a `CityCode` domain and `Boston` for a `CityName` domain, as shown in Figure 44-2.

Figure 44-2 Populated Create Domain Value Map File Dialog



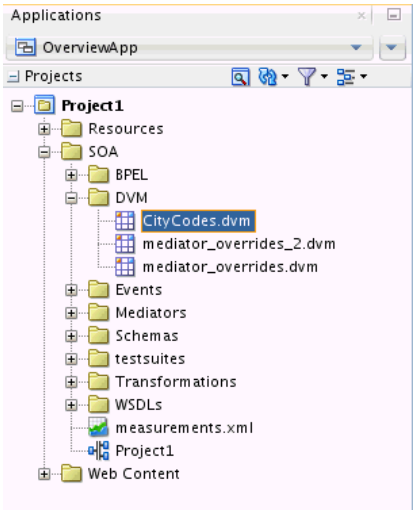
8. Click **OK**.

The Domain Value Map Editor appears with the new domain value map displayed.

What Happens When You Create a Domain Value Map

A file with the extension `.dvm` is created in the project file structure and appears in the Applications window, as shown in Figure 44-3.

Figure 44-3 A Domain Value Map File in Applications Window



All `.dvm` files are based on the schema definition (XSD) file shown in the following example:

```
<?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>
```

Editing a Domain Value Map

After you create the framework for a domain value map, you can add domains and corresponding domain values to the map using the Domain Value Map Editor.

How to Add Domains to a Domain Value Map

You can define additional domains to map, which are represented as columns in the domain value map. You can also specify whether each new domain contains values to be included in the lookups at runtime or if it is only used to qualify the mapping. Note that domain (column) names must be of the type NCName (non-colonized name), which is a valid XML element name with no colons.

To add a domain to a domain value map:

1. If the map file is not open in the Domain Value Map Editor, double-click the DVM file in the Applications window.
2. In the Map Table, click **Add** and then select **Add Domain**.

   The Create Domain dialog appears.
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**.

   Tip:

   For more information about qualifier domains and qualifier order, see Qualifier Domains and Qualifier Hierarchies..
5. In the **Qualifier Order** field, enter a number indicating the priority of the qualifier domain.

   This field is enabled only if you selected **True** in the **Qualifier** field.

Figure 44-4 Domain Value Map - Create Domain Dialog

6. Click **OK**.

   A new column appears in the Map Table.

How to Edit a Domain

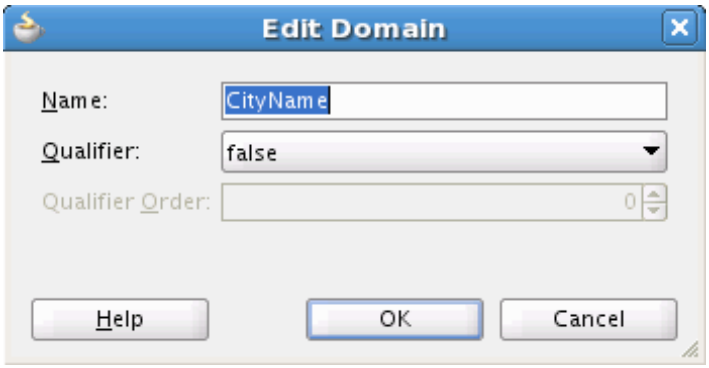
Once you add a domain to a domain value map, you can change the name, change whether it is a qualifier domain, and change the qualifier order.

To edit a domain

1. In the Domain Value Map Editor, select the name of the domain you want to modify.
2. Click **Edit Domain/Values**.

   The Edit Domain dialog appears.

Figure 44-5 Domain Value Map - Edit Domain Dialog



3. Change any of the fields on the dialog, and then click **OK**.

Note:

Domain names must be of the type NCName (non-colonized name), which is a valid XML element name with no colons.

How to Add Domain Values to a Domain Value Map

Domain values are displayed in rows in the domain value map, with each row containing the values to be mapped for each domain. You can add as many domain values as required to fully define the mapping between domains.

To add domain values to a domain value map:

1. In the Domain Value Map Editor, click **Add** and then select **Add Domain Values**.

   A new row appears beneath the existing rows in the Map Table.

2. Enter the values for each domain in the new row.

3. Repeat the above steps to create additional rows. When you are done making changes, click **Save All** on the Oracle JDeveloper toolbar.

How to Edit Domain Values

Once you add domain values to a domain value map, you can modify the values if needed.

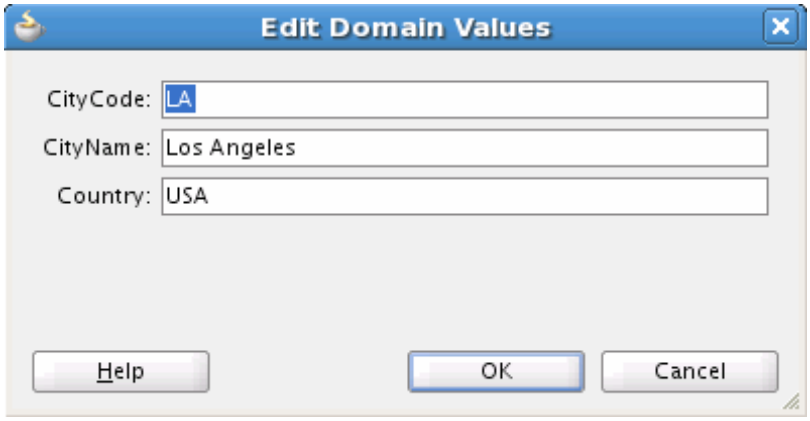
To modify domain values

1. In the Domain Value Map Editor, select the row containing the values you want to modify.

2. Click **Edit Domain/Values**.

   The Edit Domain Values dialog appears.

Figure 44-6 Domain Value Map - Edit Domain Values



3. Modify any of the fields on the dialog, and then click **OK**.

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.

Understanding Domain Value Map Functions

The `dvm:lookupValue` and `dvm:lookupValue1M` XPath functions look up a domain value map for a single value or multiple values at runtime.

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.

- The following code shows an example of `dvm:lookupValue` function syntax.

```
dvm:lookupValue(dvmMetadataURI as string, SourceColumnName as string,
 SourceValue as string, TargetColumnName as string, DefaultValue as string) as
 string
```

  The following code provides an example of `dvm:lookupValue` function use.

```
dvm:lookupValue('cityMap.dvm','CityCodes','BO', 'CityNames',
'CouldNotBeFound')
```

- The following code shows another example of `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
```

  The following code provides another example of `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.

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. The following example provides details:

```
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.

The following code shows an example of `dvm:lookupValue1M` function use.

```
dvm:lookupValue1M ('cityMap.dvm','CityCode','BO','CityName',
'CityNickName')
```

The result is shown in the following example:

```
<CityName>Boston</CityName>
<CityNickName>BeanTown</CityNickName>
```

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 a Mediator service component. Transformations are performed by using the XSLT Mapper, which appears when you create an XSL file to transform the data from one XML schema to another.

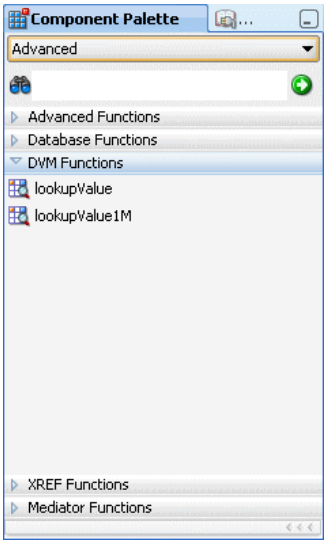
For information about the XSLT Mapper, see Creating Transformations with the XSLT Map Editor .

To use the lookupValue1M function in a transformation:

1. In the Applications window, 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 Components window, click the down arrow, and then select **Advanced**.
4. Select **DVM Functions**, as shown in Figure 44-7.

Figure 44-7 Domain Value Map Functions in the Components Window



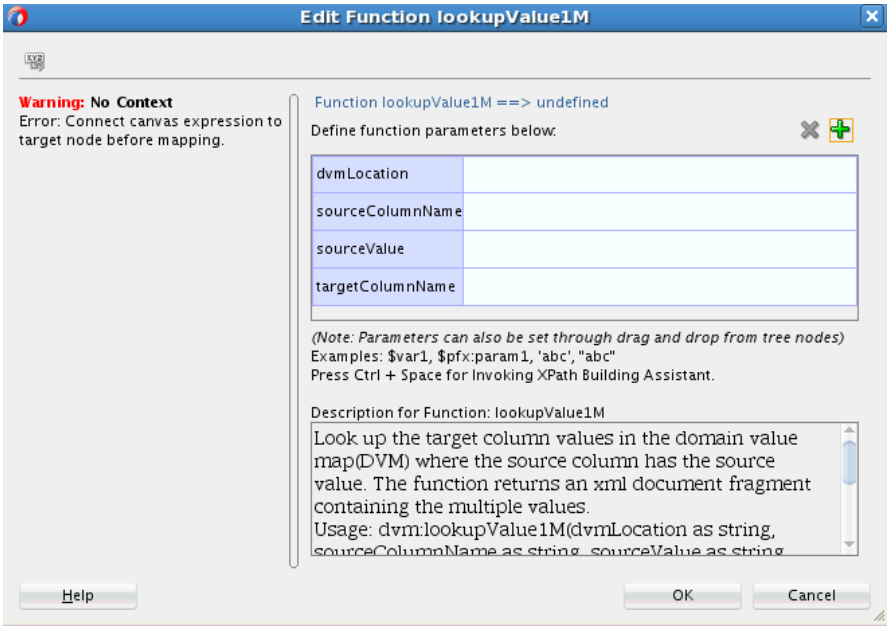
5. Drag and drop **lookupValue1M** onto the line that connects the source to the target.

   A **dvm:lookupValue1M** icon appears on the connecting line.

6. Double-click the **lookupValue1M** icon.

   The Edit Function – lookupValue1M dialog appears, as shown in Figure 44-8.

Figure 44-8 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.

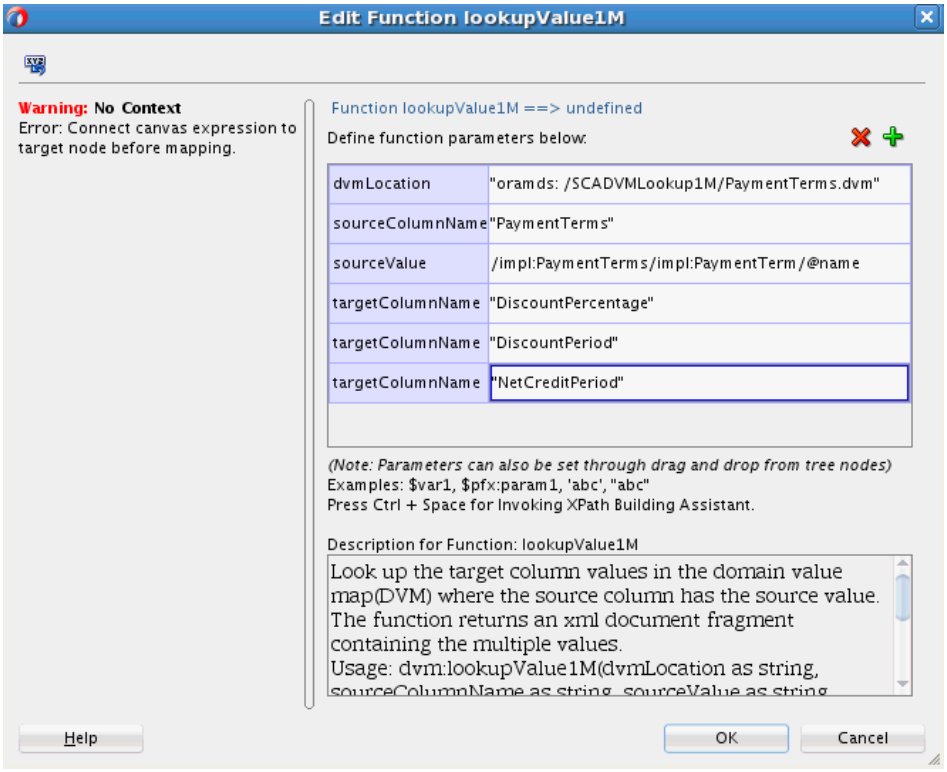
   c. In the **sourceValue** field, enter a value or press **Ctrl-Space** to use the XPath Building Assistant. Press the up and down arrow keys to locate an object in the list, and press **Enter** to select an item.

   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 44-9.

Figure 44-9 Populated Edit Function – lookupValue1M Dialog



8. Click **OK**.

   The XSLT Mapper appears with the **lookupValue1M** function icon.

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

For more information about selecting deployed domain value maps, see How to Deploy and Use Shared Data Across Multiple SOA Composite Applications in .

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 How to Assign Values.

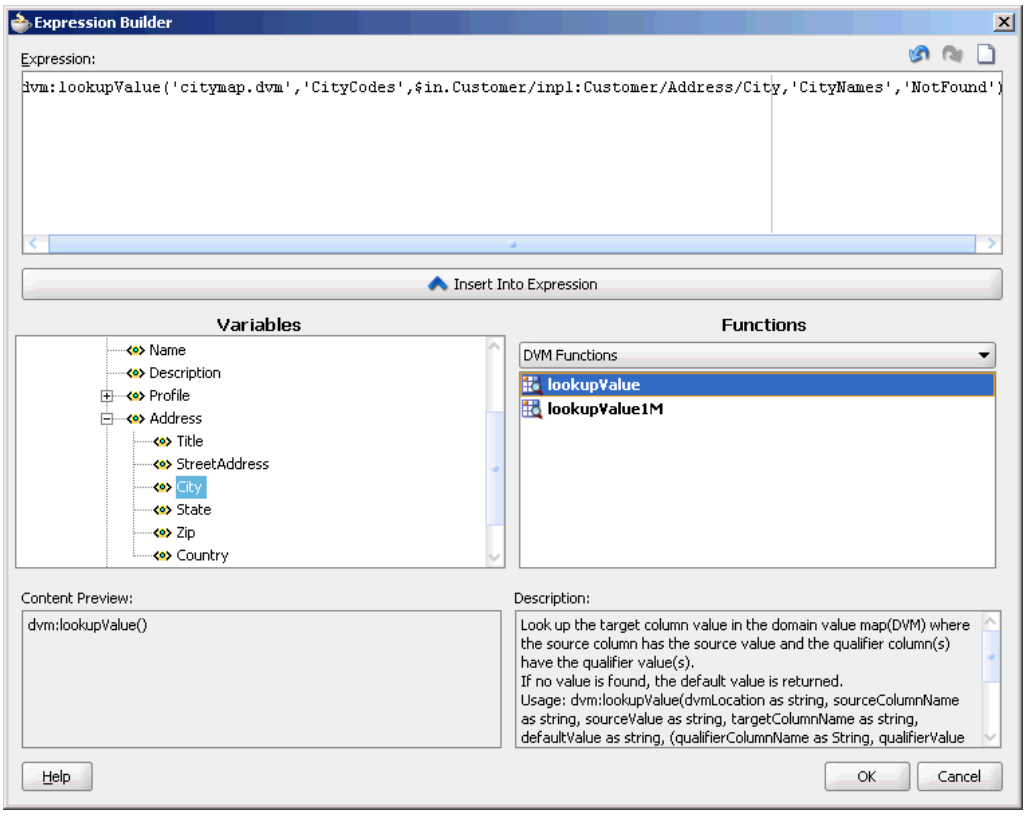
To use the lookupValue function in the Expression Builder dialog:

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 44-10, 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 44-10 Domain Value Map Functions in the Expression Builder Dialog



What Happens at Runtime

At runtime, a BPEL process service component or a Mediator service component uses the domain value map to look up appropriate values.

Creating a Domain Value Map Use Case for a Hierarchical Lookup

This section provides a tutorial for using domain value maps in a SOA composite. 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 a Mediator named ProcessOrders.
3. The ProcessOrders Mediator then transforms the message to the structure required by the adapter reference. During transformation, Mediator looks up the UnitsOfMeasure domain value map for an equivalent value of the Common domain.
4. The ProcessOrders Mediator sends the message to an external reference named WriteOrders.
5. The WriteOrders reference writes the message to a specified output directory.

To download the sample files mentioned in this section, see the `Oracle SOA Suite samples page`.

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.

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 Applications window 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**.

Task 2: How to Create a Domain Value Map

After creating an application and a project for the use case, create a domain value map.

To create a domain value map:

1. In the Applications window, 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 appears.
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 appears.
9. Click **Add** and then select **Add Column**.

   The Create DVM Column dialog appears.
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 Domain Values**.

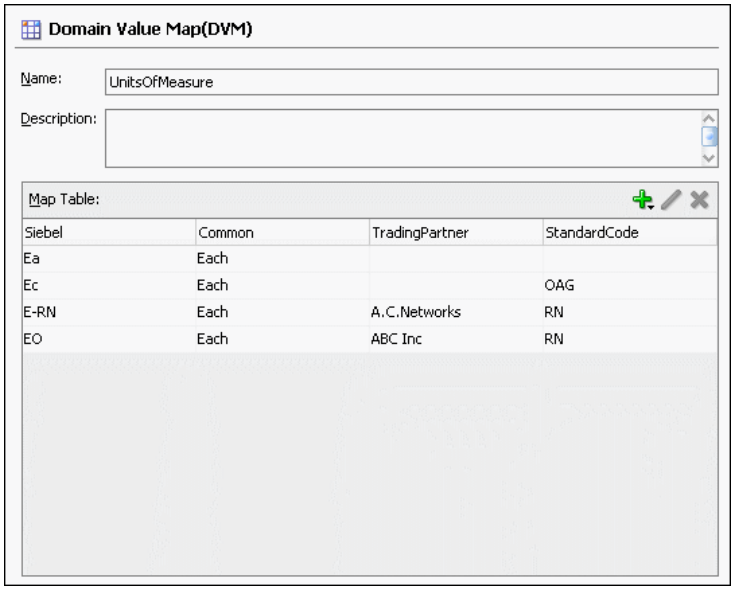
    Repeat this step to add two more rows.
15. Enter the information shown in Table 44-6 in the newly added rows of the domain value map table.

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 44-11.

Figure 44-11 UnitsOfMeasure Domain Value Map



16. From the **File** menu, select **Save All** and close the Domain Value Map Editor.

Task 3: How to Create a File Adapter Service

After creating the domain value map, 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 Components window, 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 appears.

4. In the **Service Name** field, enter `ReadOrders` and then click **Next**.

   The Operation page appears.

5. In the **Operation Type** field, select **Read File** and then click **Next**.

   The File Directories page appears.

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 appears.

8. In the **Include Files with Name Pattern** field, enter `*.xml` and then click **Next.**

   The File Polling page appears.

9. Change the **Polling Frequency** field value to **10 seconds** and then click **Next.**

   The Messages page appears.

10. Click **Search**.

    The Type Chooser dialog appears.

11. Click **Import Schema File**.

    The Import Schema File dialog appears.

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**.

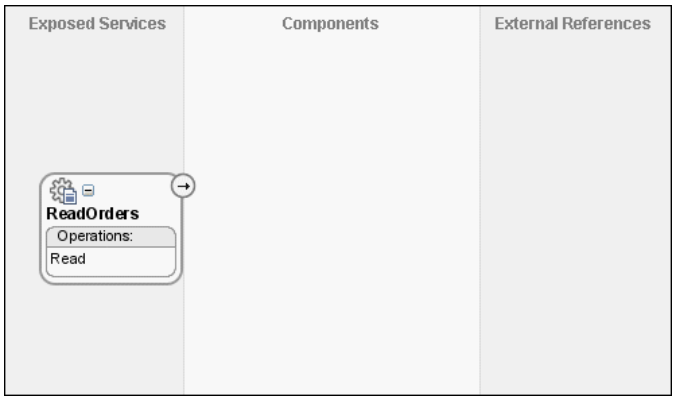
    The Finish page appears.

17. Click **Finish**.

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

    Figure 44-12 shows the **ReadOrders** service in the .

Figure 44-12 ReadOrders Service in the



Task 4: How to Create ProcessOrders Mediator Component

To create a Mediator named ProcessOrders:

1. Drag and drop a **Mediator** icon from the Components window to the **Components** section of the SOA Composite Editor.

   The Create Mediator dialog appears.

2. In the **Name** field, enter `ProcessOrders`.

3. From the **Template** list, select **Define Interface Later**.

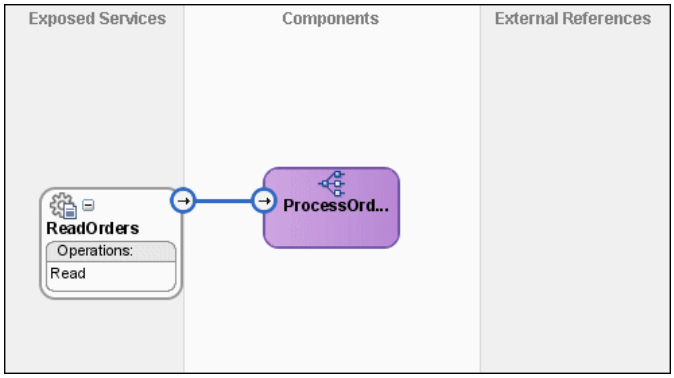
4. Click **OK**.

   A Mediator with name **ProcessOrders** is created.

5. In the , connect the **ReadOrders** service to the **ProcessOrders** Oracle Mediator, as shown in Figure 44-13.

   This specifies the file adapter service to invoke the **ProcessOrders** Mediator while reading a file from the input directory.

Figure 44-13 ReadOrders Service Connected to the ProcessOrders Mediator



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

Task 5: How to Create a File Adapter Reference

To create a file adapter reference:

1. From the Components window, select **SOA**.
2. Select **File Adapter** and drag it to the **External References** swimlane.

   The Adapter Configuration wizard Welcome page appears.
3. Click **Next**.

   The Service Name page appears.
4. In the **Service Name** field, enter `WriteCommonOrder`.
5. Click **Next**.

   The Operation page appears.
6. In the **Operation Type** field, select **Write File**.
7. Click **Next**.

   The File Configuration page appears.
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**.

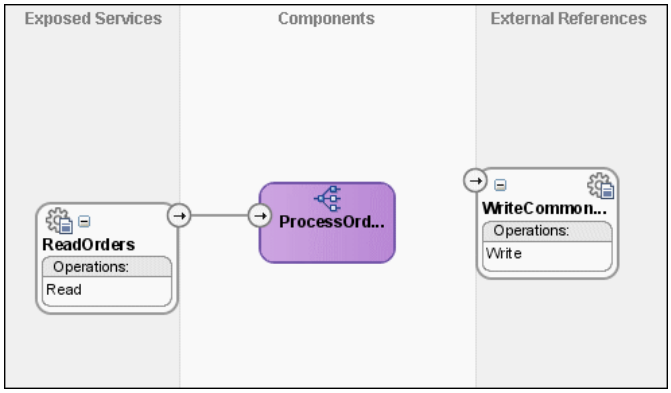
   The Messages page appears.
10. Click **Search**.

    The Type Chooser dialog appears.
11. Navigate to **Type Explorer > Project Schema Files > Order.xsd**, and then select **listOfOrder**.
12. Click **OK**.
13. Click **Next.**

    The Finish page appears.
14. Click **Finish**.

    Figure 44-14 shows the **WriteCommonOrder** reference in the .

Figure 44-14 WriteCommonOrder Reference in the



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

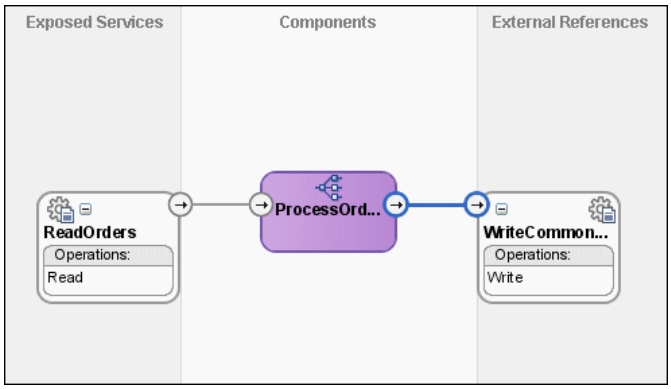
Task 6: How to Specify Routing Rules

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 44-15.

   Figure 44-15 ProcessOrders 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 appears.
4. Select **Create New Mapper File** and click **OK**.

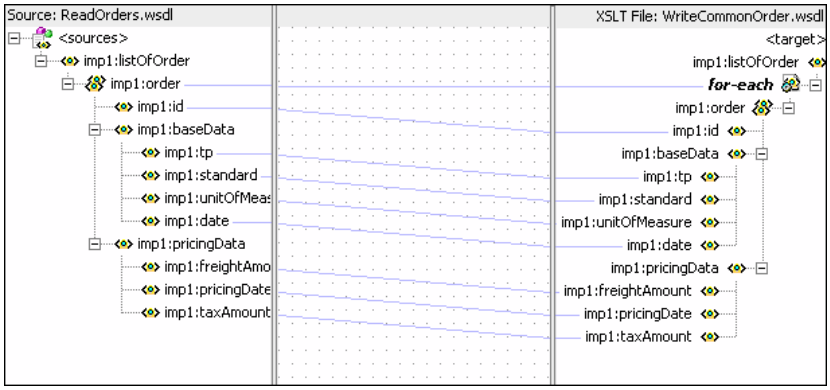
   A **listOfOrder_To_listOfOrder.xsl** file appears in the XSLT Mapper.
5. Drag and drop the **imp1:listOfOrder** source element onto the **imp1:listOfOrder** target element.

   The Auto Map Preferences dialog appears.

6. From the **During Auto Map** options, deselect **Match Elements Considering their Ancestor Names**.
7. Click **OK**.

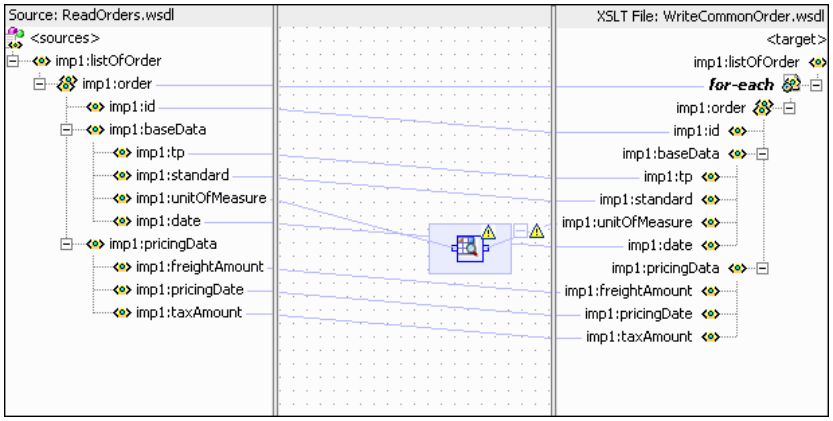
   The **listOfOrder_To_listOfOrder.xsl** file appears, as shown in Figure 44-16.

Figure 44-16 imp1:listOfOrder To imp1:listOfOrder Transformation



8. In the Components window, select **Advanced**.
9. Click **DVM Functions**.
10. Drag and drop **lookupValue** on the line connecting the **unitsOfMeasure** elements, as shown in Figure 44-17.

Figure 44-17 Adding lookupValue Function to imp1:listOfOrder To imp1:listOfOrder.xsl



11. Double-click the **lookupvalue** icon.

    The Edit Function-lookupValue dialog appears.
12. To the right of the **dvmLocation** field, click **Search**.

    The SOA Resource Lookup dialog appears.
13. Select **UnitsofMeasure.dvm** and click **OK**.
14. To the right of the **sourceColumnName** field, click **Search**.

    The Select DVM Column dialog appears.

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 appears.
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**.

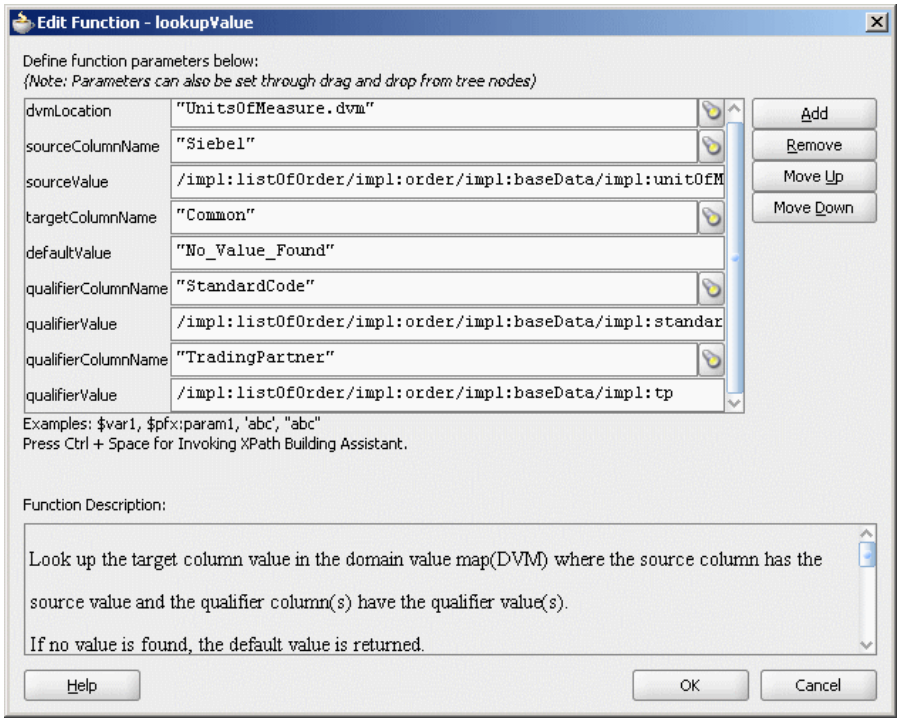
    A **qualifierValue** row is added.
23. In the **qualifierValue** field, enter the following:

    `/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 44-18.

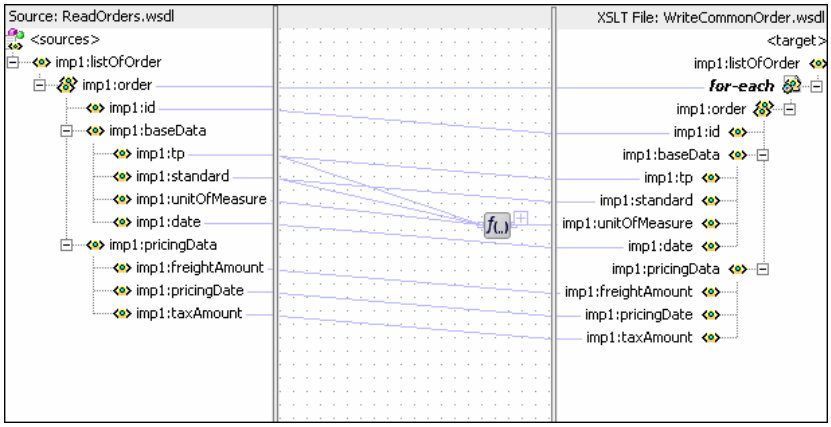
Figure 44-18 Edit Function-lookupValue Function Dialog: Hierarchical Lookup Use Case



28. Click **OK**.

The transformation appears, as shown in Figure 44-19.

Figure 44-19 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.

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 Creating an Application Server Connection.

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 How to Deploy a Single SOA Composite in .

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 How to Deploy a Single SOA Composite in .

Creating a Domain Value Map Use Case For Multiple Values

This section provides a tutorial demonstrating how to create a domain value map with multiple values to look up. 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.

To download the sample files mentioned in this section, see `Oracle SOA Suite samples page`.

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.

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 Applications window 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**.

Task 2: How to Create a Domain Value Map

After creating an application and a project for the use case, create the domain value map.

To create a domain value map:

1. In the Applications window, 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 appears.

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 appears.
11. Click **Add** and then select **Add Row**.

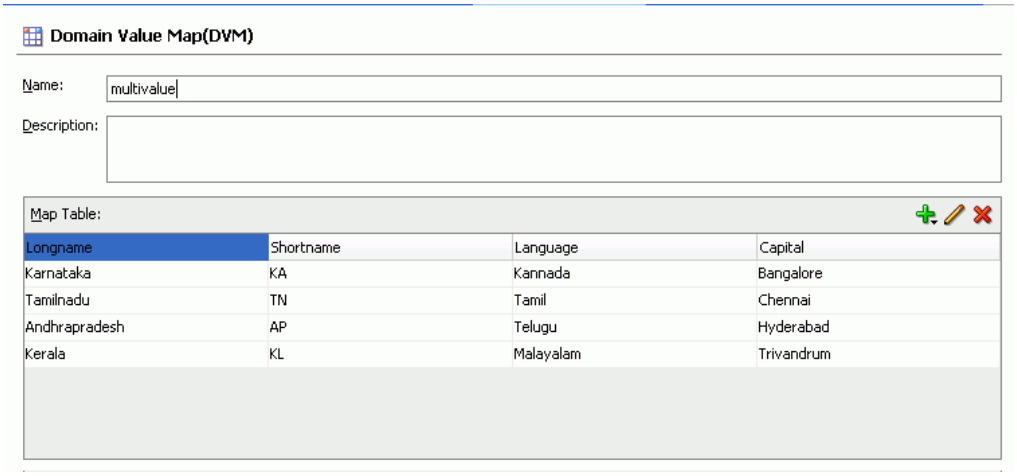
    Repeat this step to add two more rows.
12. Enter the information shown in Table 44-7 in the newly added rows of the domain value map table:

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 44-20.

Figure 44-20 Multivalue Domain Value Map



13. From the **File** menu, select **Save All** and close the Domain Value Map Editor.

Task 3: How to Create a File Adapter Service

After creating the domain value map, create a file adapter service named readFile to read the XML files from a directory.

Note:

Mediator may process the same file twice when run against Oracle RAC planned outages. This is because a file adapter is a non-XA compliant adapter. 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 Components window, 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 appears.
4. In the **Service Name** field, enter `readFile` and then click **Next**.

   The Adapter Interface page appears.
5. Click **Define from operation and schema (specified later)** and then click **Next**.

   The Operation page appears.
6. In the **Operation Type** field, select **Read File** and then click **Next**.

   The File Directories page appears.
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 appears.
9. In the **Include Files with Name Pattern** field, enter `*.xml` and then click **Next.**

   The File Polling page appears.
10. Change the **Polling Frequency** field value to **1 second** and then click **Next.**

    The Messages page appears.
11. Click **Search**.

    The Type Chooser dialog appears.
12. Click **Import Schema File**.

    The Import Schema File dialog appears.
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 appears.

18. Click **Finish**.

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

    Figure 44-21 shows the **readFile** service in the .

Figure 44-21 readFile Service in the

Task 4: How to Create the LookupMultiplevaluesMediator Mediator

To create the LookupMultiplevaluesMediator Mediator:

1. Drag and drop a **Mediator** icon from the Components window to the **Components** section of the SOA Composite Editor.

   The Create Mediator dialog appears.

2. In the **Name** field, enter `LookupMultiplevaluesMediator`.

3. From the **Template** list, select **Define Interface Later**.

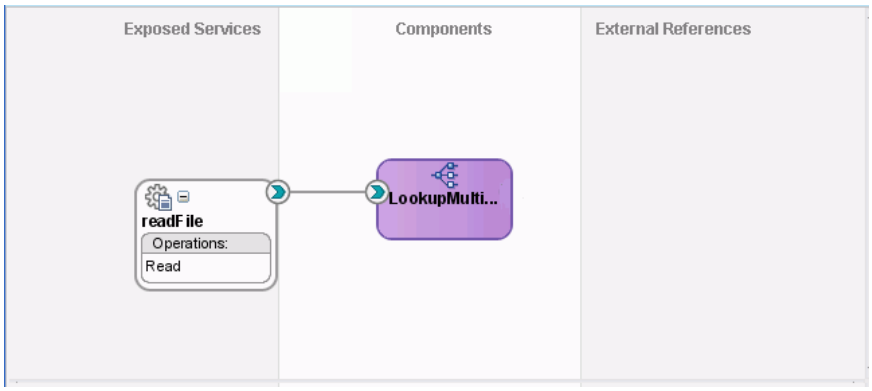
4. Click **OK**.

   An Oracle Mediator with the name **LookupMultiplevaluesMediator** is created.

5. In the , connect the **readFile** service to the **LookupMultiplevaluesMediator** Oracle Mediator, as shown in Figure 44-22.

   This specifies the file adapter service to invoke the **LookupMultiplevaluesMediator** Oracle Mediator while reading a file from the input directory.

Figure 44-22 readFile Service Connected to the LookupMultiplevaluesMediator Mediator



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

Task 5: How to Create a File Adapter Reference

To create a file adapter reference:

1. From the Components window, select **SOA**.
2. Select **File Adapter** and drag it to the **External References** swimlane.

   The Adapter Configuration wizard Welcome page appears.
3. Click **Next**.

   The Service Name page appears.
4. In the **Service Name** field, enter `writeFile` and then click **Next**.

   The Adapter Interface page appears.
5. Click **Define from operation and schema (specified later)** and then click **Next**.

   The Operation page appears.
6. Click **Next**.

   The Operation page appears.
7. In the **Operation Type** field, select **Write File**.
8. Click **Next**.

   The File Configuration page appears.
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 appears.
11. Click **Search**.

The Type Chooser dialog appears.

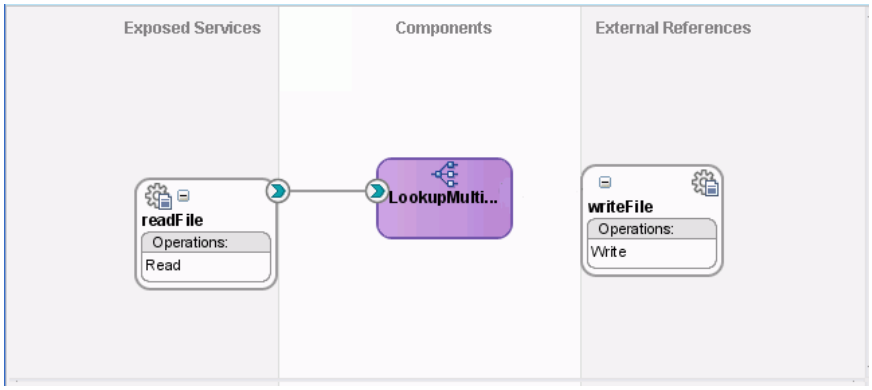
12. Navigate to **Type Explorer > Project Schema Files > output.xsd**, and then select **Root-Element**.
13. Click **OK**.
14. Click **Next.**

    The Finish page appears.

15. Click **Finish**.

    Figure 44-23 shows the **writeFile** reference in the .

Figure 44-23 writeFile Reference in



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

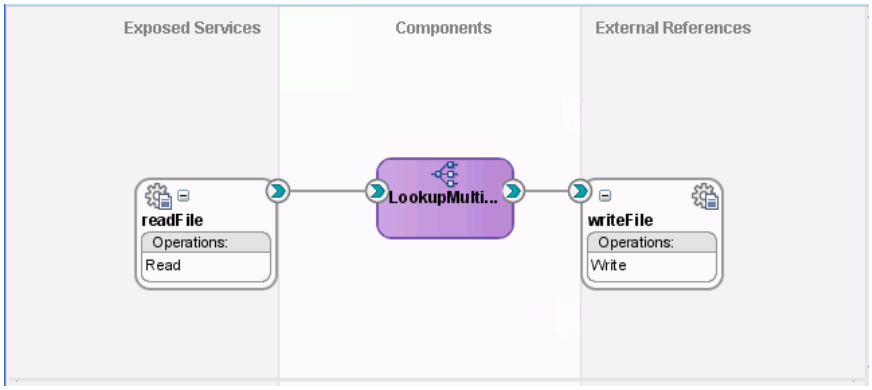
Task 6: How to Specify Routing Rules

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** Mediator to the **writeFile** reference, as shown in Figure 44-24.

Figure 44-24 LookupMultiplevaluesMediator Mediator Connected to the writeFile Reference



2. Double-click the **LookupMultiplevaluesMediator** Mediator.
3. To the right of the **Transform Using** field, click the icon.

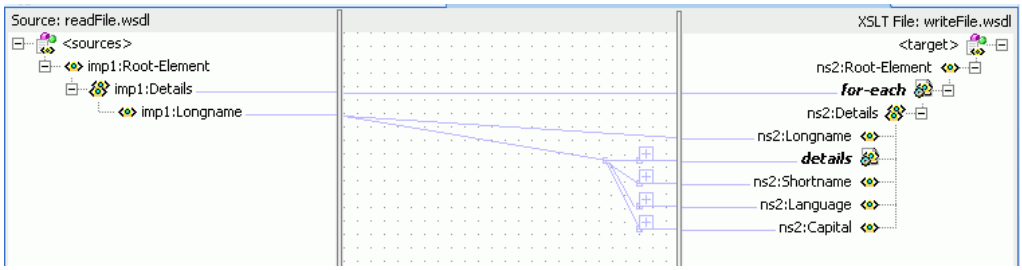
   The Request Transformation Map dialog appears.
4. Select **Create New Mapper File** and click **OK**.

   An **Input_To_Output_with_multiple_values_lookup.xsl** file appears 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 appears.
6. From the **During Auto Map** options list, deselect **Match Elements Considering their Ancestor Names**.
7. Click **OK**.

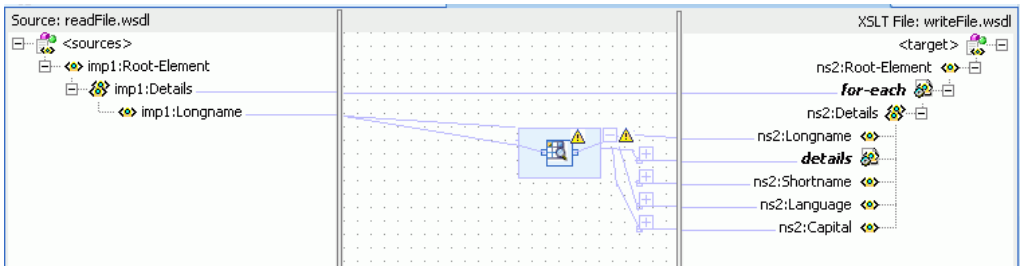
   The **Input_To_Output_with_multiple_values_lookup.xsl** file appears in the XSLT Mapper, as shown in Figure 44-25.

Figure 44-25 imp1:Root-Element To ns2:Root-Element Transformation



8. In the Components window, select **Advanced**.
9. Click **DVM Functions**.
10. Drag and drop **lookupValue1M** in the center panel, as shown in Figure 44-26.

Figure 44-26 Adding lookupValue Function to imp1:Root-Element to ns2:Root-Element



11. Double-click the **lookupvalue1M** icon.

    The Edit Function-lookupValue1M dialog appears.
12. To the right of the **dvmLocation** field, click **Search**.

    The SOA Resource Lookup dialog appears.

13. Select **multivalue.dvm** and click **OK**.
14. To the right of the **sourceColumnName** field, click **Search**.

    The Select DVM Column dialog appears.
15. Select **Longname** and click **OK**.
16. In the **sourceValue** column, enter the following:

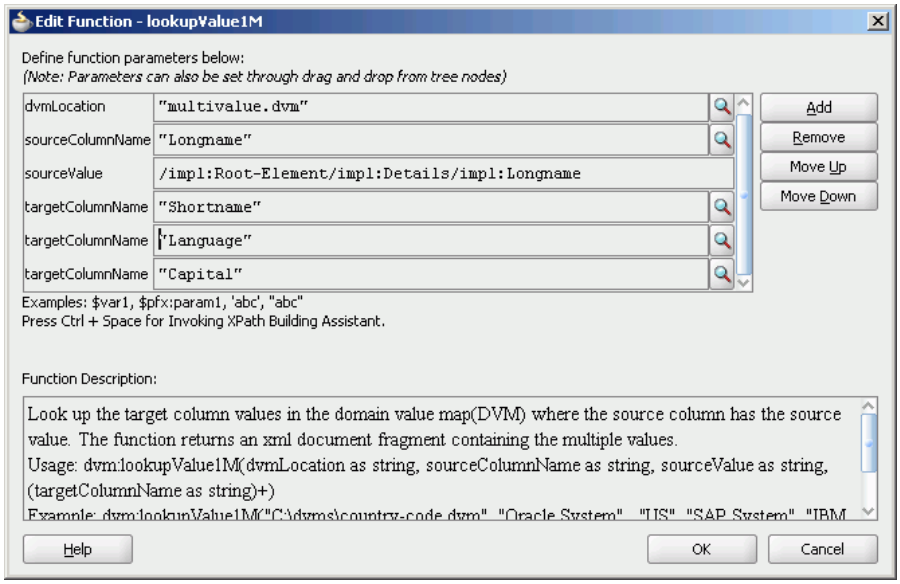
    `/imp1:Root-Element/imp1:Details/imp1:Longname`.
17. To the right of the **targetColumnName** field, click **Search**.

    The Select DVM Column dialog appears.
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 44-27.

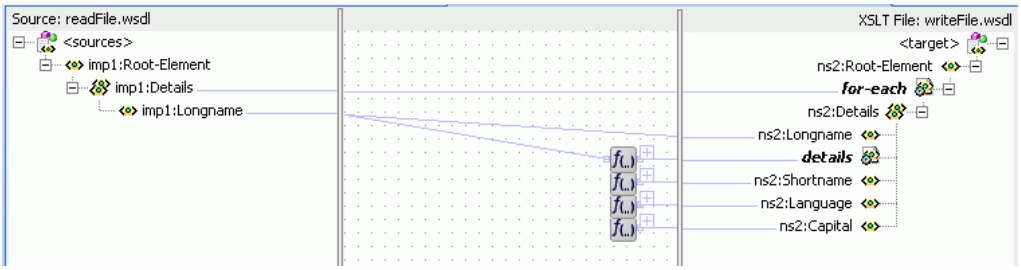
Figure 44-27 Edit Function-lookupValue Function Dialog: Multiple Value Lookup Use Case



23. Click **OK**.

    The Transformation appears, as shown in Figure 44-28.

Figure 44-28 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.

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 Creating an Application Server Connection.

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 How to Deploy a Single SOA Composite in .

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 appears.

Preloading DVM Cache for Faster First-Use

When a DVM is first called into use, the DVM gets loaded into the cache from the MDS. Subsequent lookups are faster, as the DVM is picked from the cache.

If you have a lot of records in your DVMs, you may want to preload the DVMs into the cache during server startup, so that the DVMs are readily available for first use.

You can choose to preload the DVM cache at server startup using the MBean property **LoadDVMsAtStartup** in the System MBean Browser of Oracle Enterprise Manager Fusion Middleware Control. Setting **LoadDVMsAtStartup** to **true** loads all the DVMs into the cache at server startup. The default value for **LoadDVMsAtStartup** is false.

How to Preload DVM Cache at Server Startup

To preload DVM cache at server startup:

1. Log in to Oracle Enterprise Manager Fusion Middleware Control.
2. From the **SOA Infrastructure** menu, select **SOA Administration** > **Common Properties**.
3. At the bottom of the SOA Infrastructure Common Properties page, click **More SOA Infra Advanced Configuration Properties**.
4. Click **LoadDVMsAtStartup**.
5. In the **Value** field, select `true`.
6. Click **Apply**.
7. Click **Return**.

45

Using Oracle SOA Composer with Domain Value Maps

This chapter describes how to modify domain value maps for an Oracle SOA Suite project at runtime using Oracle SOA Composer. Domain value maps let you 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.

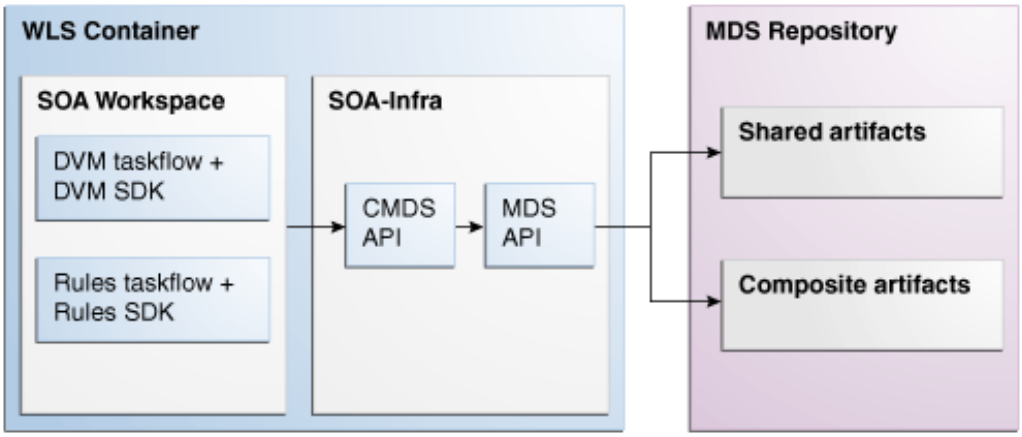
This chapter includes the following sections:

For more information about domain value maps, see Working with Domain Value Maps .

Introduction to Oracle SOA Composer

Oracle SOA Composer is an EAR file that is installed as part of the Oracle SOA Suite installation. Oracle SOA Composer enables you to manage deployed domain value maps during runtime without needing to redeploy the project that uses the 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 lets you access a domain value map from the Metadata Service (MDS) repository.

Figure 45-1 Oracle SOA Composer High-Level Deployment Topology



How to Log in to Oracle SOA Composer

To log in to Oracle SOA Composer:

1. Enter the following URL in your web browser:

   `http://hostname:port/soa/composer`

   The Oracle SOA Composer Login page appears, as shown in Figure 45-2.

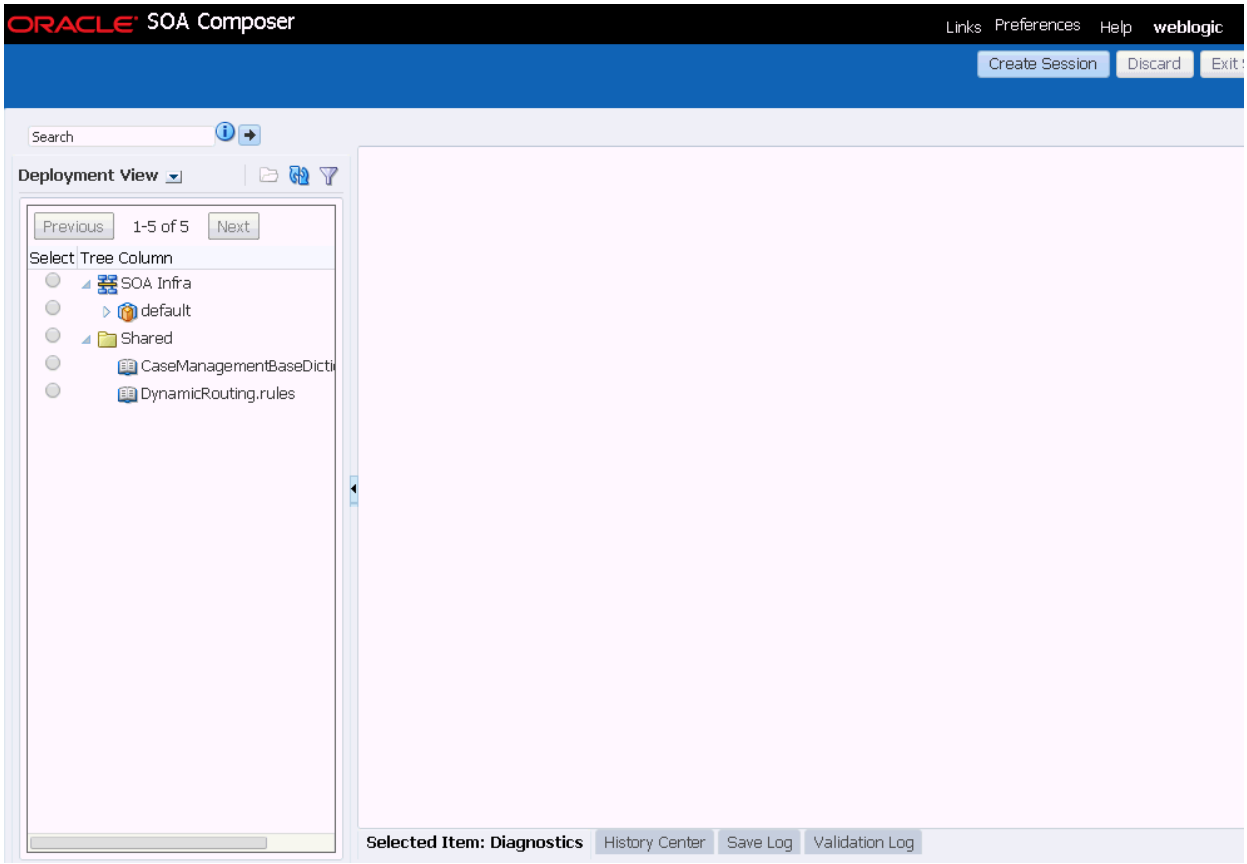
Figure 45-2 Oracle SOA Composer Login Page

2. In the **Username** field, enter your user name.
3. In the **Password** field, enter your password.

4. Click **Login**.

After you log in to Oracle SOA Composer, the Oracle SOA Composer home page appears, 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 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 *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

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.

How To View Domain Value Maps at Runtime

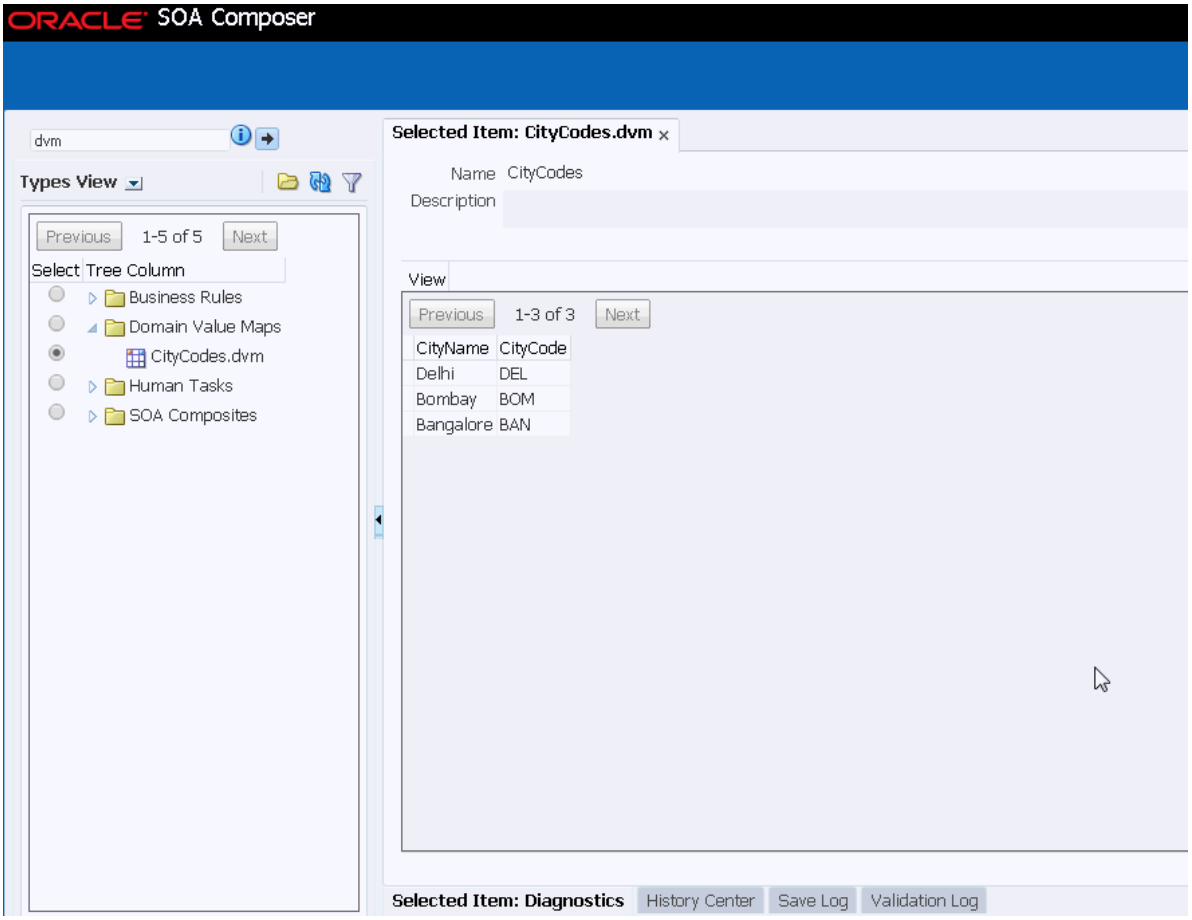
To view domain value maps at runtime:

1. Select **Types View** in the Applications window panel on the left.

2. Expand **Domain Value Maps** folder by clicking the right arrow icon before it.
3. Select the domain value map file (.dvm) that you want to view or edit.
4. Click the **Open** icon to open the domain value map.
5. From the **Open** menu, select **Open DVM**. The DVM details appear in view mode.

   Figure 45-4 shows a sample domain value map in SOA Composer.

Figure 45-4 Viewing a Domain Value Map in SOA Composer



Note:

To get a direct link to the selected domain value map, click **Bookmark**.

Editing Domain Value Maps at Runtime

You can edit domain value maps while the applications using the domain value map are running.

Note:

When you update a DVM using SOA Composer, the DVM cache also gets updated with the updated DVM.

How to Edit Domain Value Maps at Runtime

By default, domain value maps open in view mode. Once you change to edit mode, you can modify row information. When you finish making changes, be sure to save and commit them as described in Publishing Changes at Runtime.

Changing to Edit Mode

To change to edit mode:

1. Open the domain value map for viewing, as described in How To View Domain Value Maps at Runtime.
2. Click **Create Session** in the top right section of the SOA Composer window. If you have a previously active session, you must click **Edit Session.**

   The domain value map opens in edit mode.

Adding Rows

To add rows:

You can add rows by performing the following steps:

1. Click **Add Domain Values**.

   The Add Domain Values dialog appears.
2. Enter values and click **OK**.

   The entered values are added to the domain value map.
3. Click **Save** in the top right section of the SOA Composer.

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 appears.
3. Edit the values as required and click **OK**.
4. Click **Save** in the top right section of SOA Composer.

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**.
3. Click **Save** in the top right section of SOA Composer.

Publishing Changes 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.

You must publish the changes you make to have them picked up by the runtime and be saved permanently to the MDS repository. In a session, you can also save your changes without publishing them. In such a case, the domain value map remains in the saved state. You can reopen the domain value map and publish the changes later.

How to Publish Changes at Runtime

To publish changes at runtime:

1. Click **Publish** in the top right section of SOA Composer. A confirmation dialog appears.
2. Enter an optional description for the changes made in the session. Click **OK.**

How to Discard Changes at Runtime

You can also choose to discard any changes made to the DVM in the session.

To discard changes at runtime:

1. Click **Discard** in the top right section of SOA Composer. A confirmation dialog appears.
2. Click **OK** to discard changes made in the session. This includes any changes that you might have saved to the sandbox.

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 dialog asking you to confirm whether you want to go ahead with the edit.

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.

Part VIII

Completing Your Application

This part describes how to complete design of your application.

This part contains the following chapters:

46

Enabling Security with Policies and Message Encryption

This chapter describes how to attach policies to binding components and service components during design-time in SOA composite applications and encrypt and decrypt specific fields of messages. Policies apply security to the delivery of messages. This chapter also describes how to override policy configuration property values.

This chapter includes the following sections:

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 46-1 describes the supported policy categories.

Table 46-1 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.

Table 46-1 (Cont.) Supported Policy Categories

Category	Description
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/no_reliable_messaging_policy

  Supports the disabling of reliable messaging configured at a higher scope

- oracle/no_wsrm_policy

  Supports the disabling of a globally attached Web Services Reliable Messaging policy

- oracle/reliable_messaging_policy

  Supports the enabling of Web services reliable messaging

- 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

For more information about available policies, details about which ones to use in your environment, and global policies, see *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

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 *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

How to Attach Policies to Binding Components and Service Components

To attach policies to binding components and service components:

1. In the SOA Composite Editor, right-click a service binding component or reference binding component.

2. Select **Configure SOA 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.

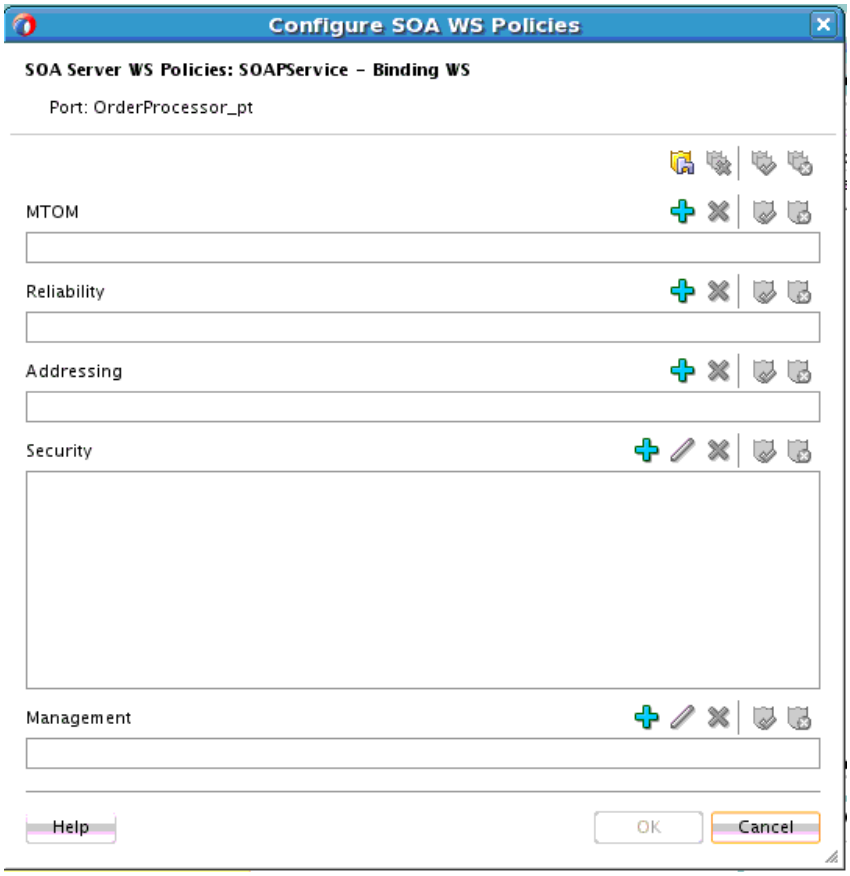
   - **For Callback**: (only for interactions with asynchronous processes)

     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 46-1 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 46-1 Configure SOA WS Policies Dialog

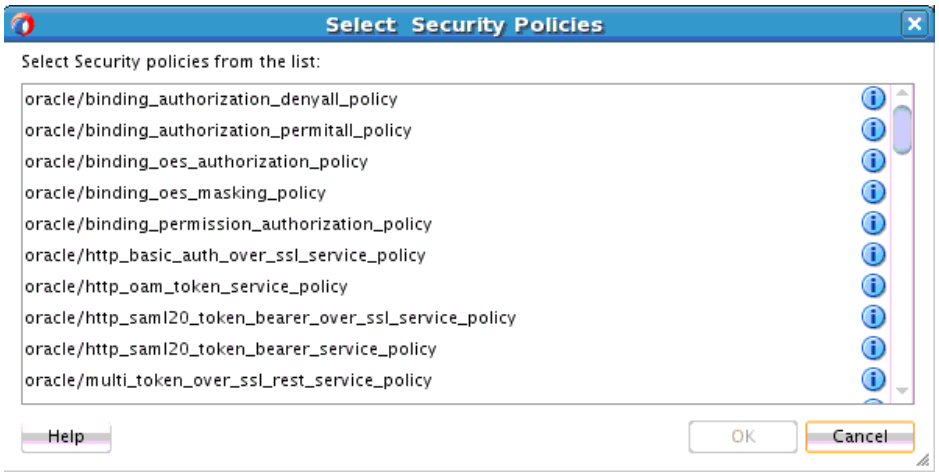


4. Click the **Add** icon next to the type of policy to attach:

   - **MTOM**
   - **Reliability**
   - **Addressing**
   - **Security**
   - **Management**

   For this example, **Security** is selected. The dialog shown in Figure 46-2 is displayed.

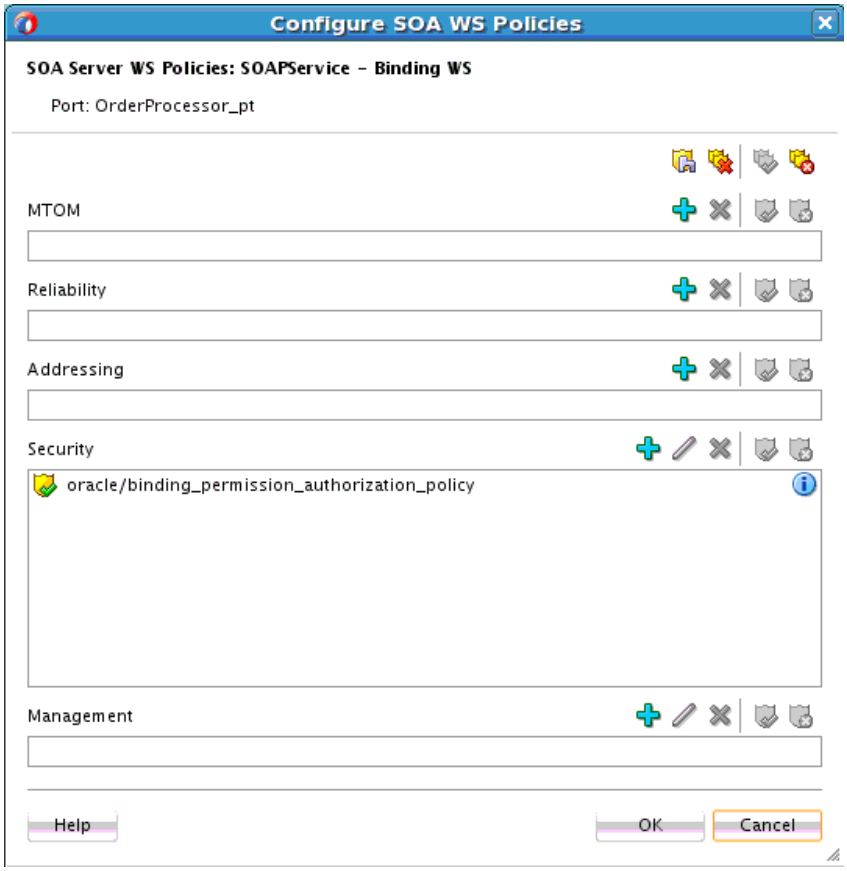
Figure 46-2 Security Policies



5. Click the icon to the right of the 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 46-3. The attached security policy displays in the **Security** section.

Figure 46-3 Attached Security Policy



8. If necessary, add additional policies.

   You can temporarily disable a policy by clicking the **Disable selected policies** icon. Figure 46-4 provides details. This action does *not* detach the policy.

   Figure 46-4 Disable Selected Policies Icon

9. To enable the policy again, click the **Enable selected policies** icon to the left.

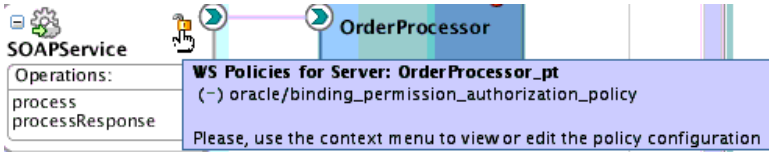
10. To detach a policy, click the **Delete** icon.

11. When complete, click **OK** in the Configure SOA WS Policies dialog.

    You are returned to the SOA Composite Editor.

12. Place your cursor over the icon on the service binding component to display details about the attached policy. Figure 46-5 provides details.

    Figure 46-5 Policy Description Icon



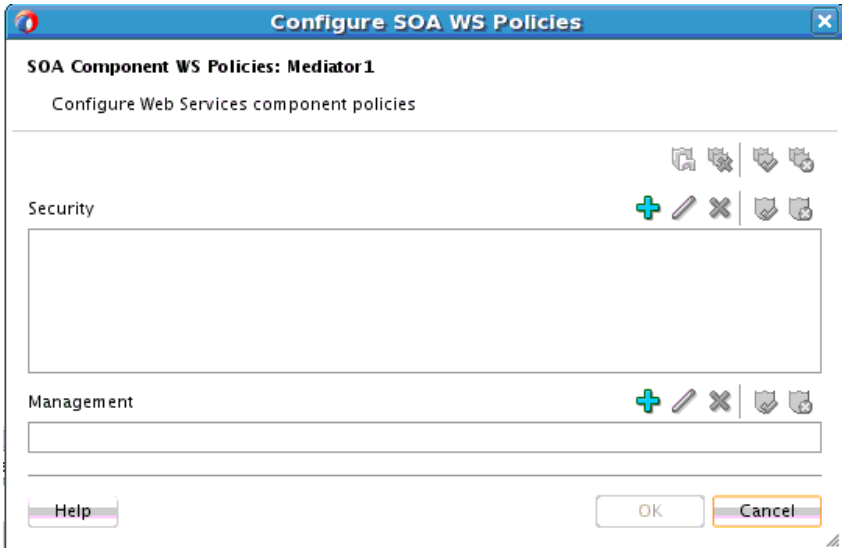
To attach a policy to a service component:

1. Right-click a service component.

2. Select **Configure SOA WS Policies**.

   The Configure SOA WS Policies dialog shown in Figure 46-6 appears.

   Figure 46-6 Configure SOA WS Policies Dialog



3. Click the **Add** icon next to 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 *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

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.

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.

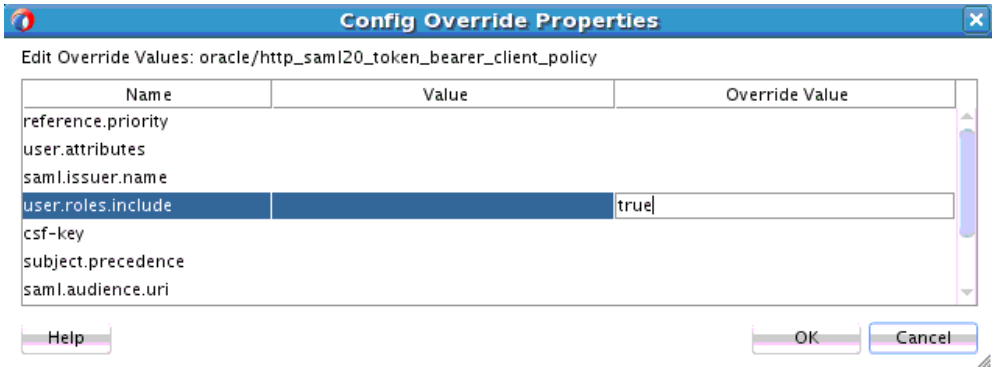
   The **Edit** icon is enabled for both sections. Figure 46-7 provides details.

Figure 46-7 Client Policy Selection

3. Click the **Edit** icon.

4. In the O**verride Value** column, enter a value to override the default value shown in the **Value** column. Figure 46-8 provides details.

Figure 46-8 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.

For more information about overriding policy settings, see *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

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.

To override server configuration property values:

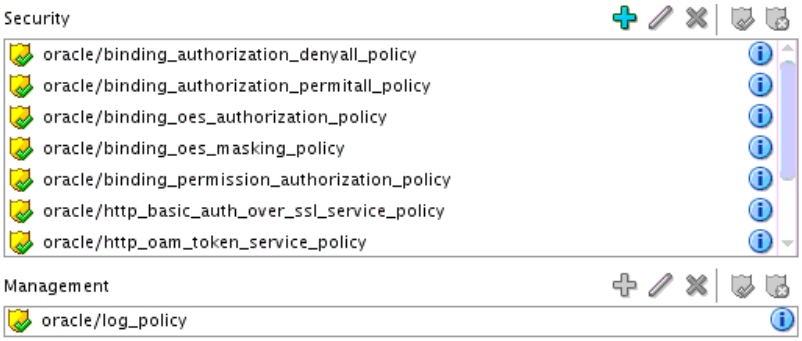
1. Right-click one of the following binding components:

   - 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.

   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 46-9 provides details.

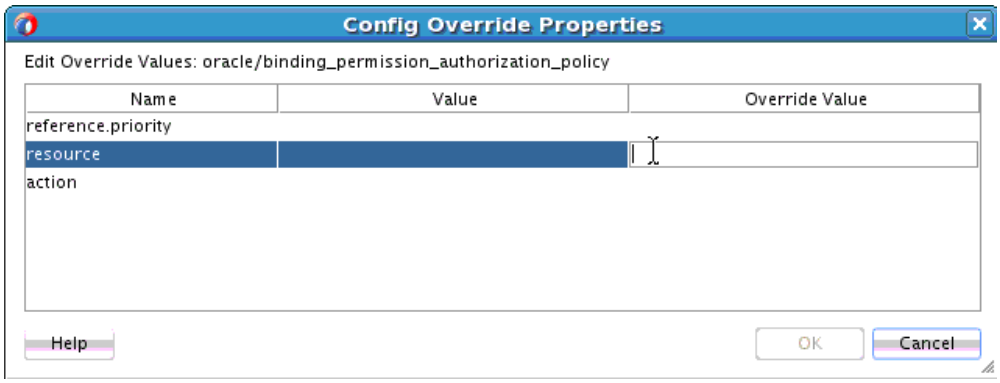
Figure 46-9 Server Policy Selection



3. Select an attached policy that permits you to override its value, and click the **Edit** icon.

4. In the O**verride Value** column, enter a value to override the default value shown in the **Value** column. Figure 46-10 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 46-10 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.

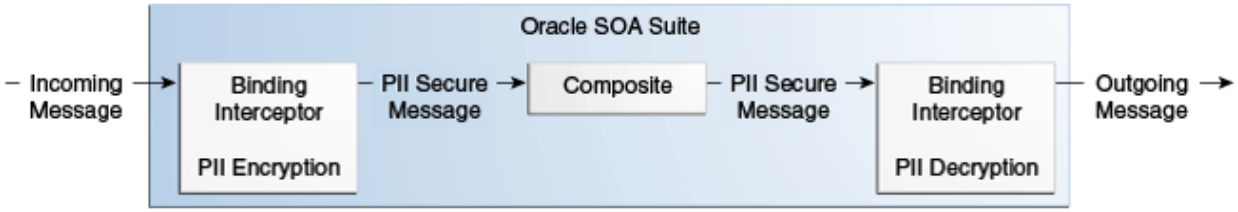
   For more information about overriding policy settings, see *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

Encrypting and Decrypting Specific Fields of Messages

You can encrypt and decrypt fields of a message to protect sensitive data (known as personally identifiable information (PII)) flowing in web services and JCA adapters in Oracle SOA Suite and Oracle Service Bus. This feature provides for the obfuscation of certain fields (for example, SSNs) to prevent this data from appearing in administration consoles in clear text.

Figure 46-11 shows an incoming message being encrypted when entering the SOA composite application in a service binding component and an outgoing message being decrypted when exiting the SOA composite application in a reference binding component. Messages outside the composite can be protected with other message protection policies (WS-Security/SSL).

Figure 46-11 Message Encryption and Decryption in a SOA Composite Application



The following code shows an example of an unencrypted message. The PII fields are `name` and `driversLicense`.

```
<person>
  <name>John</name>
  <driversLicense>B1234</driversLicense>
  <ssn>123-456-789</ssn>
</person>
```

The following code shows an example of the encrypted message with the `name` and `driversLicense` fields in encrypted format.

```
<person>
  <name>John</name>
 <driversLicense>encrypted:fdslj[lmsfwer09fsn;keyname=pii-csf-key</driversLicense>
  <ssn>encrypted:gdf45md%mfsd103k;keyname=pii-csf-key</ssn>
</person>
```

The encryption format is as follows:

```
encrypted:<CIPHER_TEXT>;keyname:<CSF_KEY_NAME>
```

Note:

If both a PII policy and authorization policy are attached to a SOA composite application, the authorization policy is executed before the PII policy. This is because the PII policy may encrypt the field used for authorization.

If the authorization policy is attached to a component and it requires an already-encrypted field, authorization fails.

How to Encrypt and Decrypt Specific Fields of Messages

Note:

- You must decrypt PIIs when an encrypted message leaves the composite. If you attach a PII policy to a service binding component and do not attach a PII policy to a reference binding component, PIIs in the outbound message are not decrypted. This is not a recommended practice, and you receive a runtime error.
- PIIs encrypted in one SOA composite application cannot be decrypted in another SOA composite application.

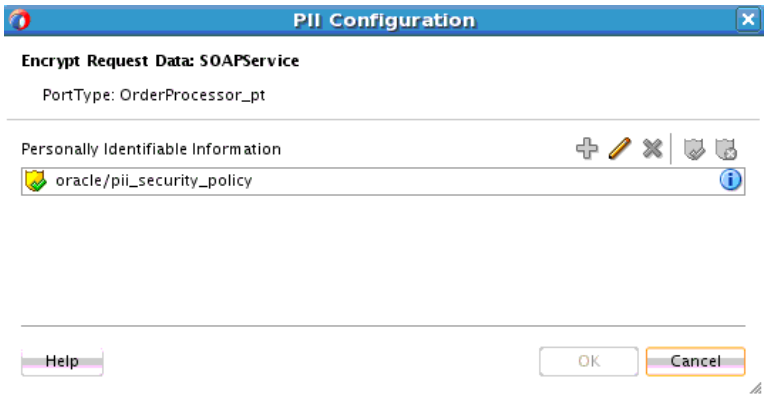
To encrypt and decrypt specific fields of messages:

1. Right-click a service binding component, and select **Protect Sensitive Data** > **Encrypt Request Data**.

   The PII Configuration dialog is displayed, as shown in Figure 46-12.

   You must now perform the initial encryption on the incoming message.

Figure 46-12 PII Configuration Dialog for Encryption



2. Click the **Edit** icon to identify the elements in the schema to encrypt.

   The **Input** tab of the Select fields to encrypt dialog is displayed.

3. Click the **Add** icon to create an XPath expression that identifies the fields of the request message to encrypt (for example, a user's name, credit card number, or social security number).

4. Click the **CSF** tab.

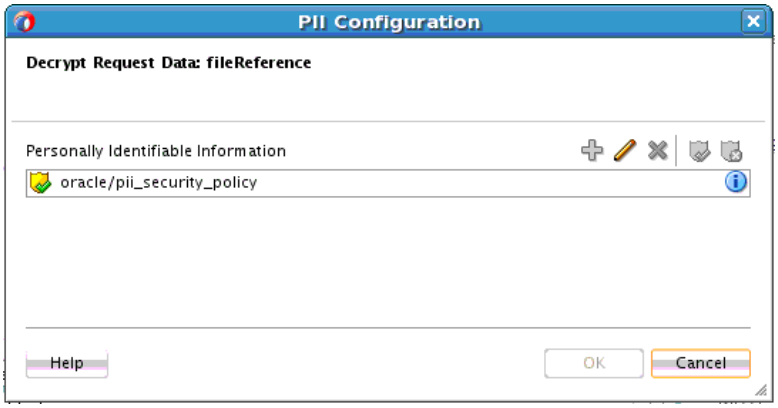
5. Select the credential store framework (CSF) key to use. The credential store is used for the secure storage of credential keys.

   After encryption is complete, the message proceeds through the service components of the SOA composite application.

   When the message reaches a reference binding component and is ready to exit the SOA composite application, you must decrypt the encrypted message.

6. Right-click the reference binding component, and select **Decrypt Sensitive Data**. Figure 46-13 provides details.

Figure 46-13 PII Configuration Dialog for Decryption



7. Click the **Edit** icon.

   The **Input** tab of the Select fields to decrypt dialog is displayed. For asynchronous processes, there are two steps: one for the input message and one for the output message.

8. Click the **Add** icon to invoke the Expression Builder dialog for creating an XPath expression that identifies the fields to decrypt (for example, a credit card number or driver's license field).

9. Click **OK** when complete.

   After configuring composites with oracle/pii_security_policy, you must add keys and user credentials to the credential store.

10. Use the `createCred` WLST command to create entries in the `oracle.wsm.security` credential map for any `csf-key` user credentials.

```
connect("weblogic","password","t3://myAdminServer.example.com:7001")

wls:/DefaultDomain/serverConfig> createCred(map="oracle.wsm.security",
key="pii-csf-key", user="weblogic", password="password", desc="Key for
pii_security_policy")
```

   If you do not perform this task, the following error occurs:

```
oracle.wsm.security.SecurityException: WSM-00016 : The
username/password credentials or certificates pii-csf-key are missing.
```

47

Deploying SOA Composite Applications

This chapter describes the deployment life cycle of SOA composite applications. It describes how to deploy single composites, multiple composites, and composites using shared data such as WSDLs, XSDs, and other file types with Oracle JDeveloper and the `ant` scripting tool, and create configuration plans for moving SOA composite applications to and from different environments. Deployment prerequisite, packaging, preparation, and configuration tasks are also described. A reference to documentation for deploying with the Oracle WebLogic Scripting Tool (WLST) utility is also provided.

This chapter includes the following sections:

See *Administering Oracle SOA Suite and Oracle Business Process Management Suite* for instructions about deploying SOA composite applications from Oracle Enterprise Manager Fusion Middleware Control and *WLST Command Reference for SOA Suite* for instructions about deploying SOA composite applications with the WLST utility.

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 *Administering Oracle Fusion Middleware*.

Deployment Prerequisites

This section describes the basic prerequisites required for creating and deploying a SOA composite application.

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 *Installing and Configuring Oracle SOA Suite and Business Process Management* and *Creating Schemas with the Repository Creation Utility*.

If you use the Oracle SOA Suite Quick Start installation in a development environment, the schema is automatically created in the Java database for you. For more information, see *Installing SOA Suite and Business Process Management Suite Quick Start for Developers*.

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 *Installing and Configuring Oracle SOA Suite and Business Process Management*.

If you install the Oracle SOA Suite Quick Start, you can configure the Integrated WebLogic Server's default domain in Oracle JDeveloper. For information, see Section "Configuring a Domain" of *Installing SOA Suite and Business Process Management Suite Quick Start for Developers*.

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 *High Availability Guide*.

Understanding the Packaging Impact

You can separately package all required artifact files within the 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 **Generate SAR File** option. For more information, see Deploying the Profile.

- `ant` scripts

  Use the `ant-sca-package` script to package your artifacts. For more information, see How to Use ant to Package a SOA Composite Application into a Composite SAR File.

- WLST commands

  Use the `sca_package` script to package your artifacts. For more information, see *WLST Command Reference for SOA Suite*.

- Maven plug-in

  Use the Maven plug-in to compile, package, deploy, test, and undeploy a SOA composite application in a Maven environment. For more information, see Using the Development Maven Plug-In.

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 How to Deploy a Single SOA Composite in and How to Deploy Multiple SOA Composite Applications in .

- A ZIP file of multiple SOA composite applications that share metadata with one another

  You can deploy and use shared data across SOA composite applications. Shared data is deployed to the SOA Infrastructure on the application server as an Oracle Metadata Services (MDS) Repository archive JAR file. The archive file contains all shared resources. For more information, see How to Deploy and Use Shared Data Across Multiple SOA Composite Applications in .

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 Oracle B2B agreements, Oracle Web Service Manager (OWSM) policies, and human workflow task flows.

- Shared data such as WSDL and XSD files. All shared data is deployed to an existing SOA Infrastructure partition on the server. This data 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`.

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.

How to Create Data Sources and Queues

A Java Database Connectivity (JDBC) data source is an object bound to the Java Naming and Directory Interface (JNDI) tree that includes a pool of JDBC connections. Applications can look up a data source in 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 47-1 require JDBC data sources and queues to be configured before deployment.

Table 47-1 Oracle JCA Adapter Tasks

Adapter	Configuration Task	See Section...
Database adapter	JDBC data source	"Deployment" of *Understanding Technology Adapters*
AQ adapter	JDBC data source	"Configuring the Data Sources in the Oracle WebLogic Server Administration Console" of *Understanding Technology Adapters*
JMS adapter	Queue	"Using the Adapter Configuration Wizard to Configure Oracle JMS Adapter" of *Understanding Technology Adapters*

Script for Creation of JMS Resource and Redeployment of JMS Adapter

The following example 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.

```
# 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"
```

For information about JMS queues and topics and connection factories, see Section "Configuring Basic JMS System Resources" of *Administering JMS Resources for Oracle WebLogic Server*.

Script for Creation of the Database Resource and Redeployment of the Database Adapter

The following example 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.

```
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"
```

For information about JDBC data sources, see Section "Configuring JDBC Data Sources" of *Administering JDBC Data Sources for Oracle WebLogic Server*.

How to Create 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 *Understanding Technology Adapters*.

How to Enable 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 Oracle iPlanet. For more information, see *Administering Oracle SOA Suite and Oracle Business Process Management Suite*. 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 *Administering 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 *Administering Web Services*.

How to Set the Business Flow Instance Name or Composite Instance Name at Design Time

You can set the business flow instance name or composite instance name of a SOA composite application during design time for Oracle Mediator and Oracle BPEL Process Manager. The name appears in the **Name** column on the Flow Instances page of a SOA composite application in Oracle Enterprise Manager Fusion Middleware Control. When you specify a search criteria on the Flow Instances page of a SOA composite application, a partition, or the SOA Infrastructure in Oracle Enterprise Manager Fusion Middleware Control, you can specify this name in the **Name** field.

Setting the Business Flow Instance Name in Oracle Mediator

To set the business flow instance name in Oracle Mediator:

Use the XPath expression function `oraext:setFlowInstanceTitle()` in an assign activity. For example:

```
<assign>
   <copy
      target="$out.property.tracking.setFlowInstanceTitle"
      expression="oraext:setFlowInstanceTitle("sample")"
      xmlns:med="http://schemas.oracle.com/mediator/xpath"/>
</assign>
```

Setting the Business Flow Instance Name in a BPEL Process

A business flow instance corresponds to an end-to-end business transaction. Business flows consist of a single SOA composite application or multiple SOA composite applications connected together to fulfill a specific business process.

To set the business flow instance name in a BPEL process:

1. Use the Java BPEL `exec` extension `bpelx:exec`. This extension includes the built-in method `setFlowInstanceTitle(String title)` for setting the business flow instance name.

   For more information about business flow instances, see Chapter "Tracking Business Flow Instances" of *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

Setting the Composite Instance Name in a BPEL Process

The `setCompositeInstanceTitle` method is provided for backward compatibility. The composite instance name is different from the business flow instance name. More than one composite instance can participate in a single business flow instance. There is a one-to-many relationship between the flow instance name and the composite instance name.

To set the composite instance name in a BPEL process:

1. Use the Java BPEL `exec` extension `bpelx:exec`. This extension includes the built-in method `setCompositeInstanceTitle(String title)` for setting the instance name.

   For more information, see *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

How to Deploy Trading Partner Agreements and Task Flows

If you are using Oracle B2B or a human task, you must perform additional setup tasks.

To deploy trading partner agreements and task flows:

- 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 *User's Guide for Oracle B2B*.

- Deploying the task flow

  You must deploy the task flow to use it in Oracle BPM Worklist. For more information, see Deploying the Profile.

How to Create an Application Server Connection

To deploy a SOA composite application that does not share data with another composite, use the Create Application Server Connection wizard to create an application server connection. For more information, see Creating an Application Server Connection.

How to Create a SOA-MDS Connection

To deploy a SOA composite application that shares data with other composites, use the Create SOA-MDS Connection wizard to create a connection to a database-based

Oracle MDS Repository server. For more information, see Creating a SOA-MDS Connection.

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 Resources window, the file may not load correctly. Only open a composite from the Applications window.

For information about the Oracle MDS Repository, see *Administering Oracle Fusion Middleware*.

Customizing Your Application for the Target Environment Before 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.

How to Use Configuration Plans to Customize 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. The configuration plan is created in either Oracle JDeveloper or with 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.

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
  - 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. To modify any XSL, use the XSLT Map Editor. 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 Map Editor in Oracle JDeveloper when moving to and from test, development, and production environments. This ensures that the XSLT Map Editor opens without any issues in design time. However, leaving the references unchanged does not impact runtime behavior. For more information about transformations and the XSLT Map Editor, see Creating Transformations with the XSLT Map Editor .

- 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 Deploying the Profile.
  - `ant` scripts

    For more information, see How to Use ant to Deploy a SOA Composite Application.
  - WLST commands

    For more information, see *WLST Command Reference for SOA Suite*.
- 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.

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 the following example:

```
<composite .....>
  <import namespace="http://example.com/hr/"
 location="http://myserver17.us.example.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.example.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 the following example:

```
<?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.

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.

How to Use a Configuration Plan when Creating Environment-Independent Processes

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.

How to Create a Configuration Plan in Oracle JDeveloper

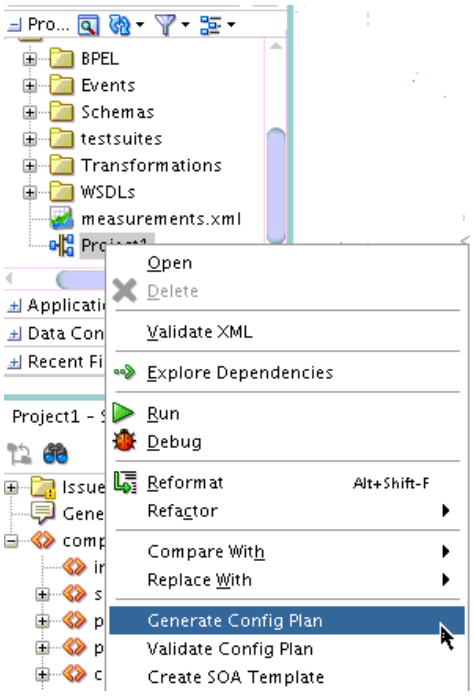
This section describes how to create and use a configuration plan. In particular, this section describes the following:

- 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:

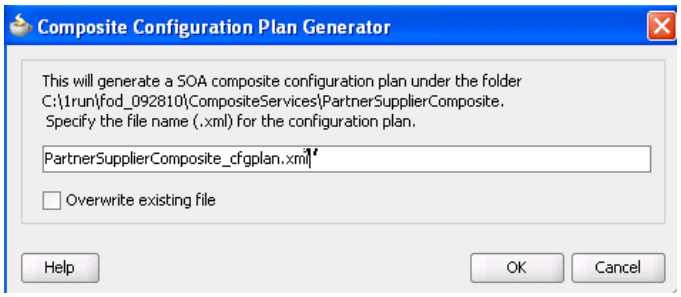
1. Open Oracle JDeveloper.
2. In the Applications window, right-click the ***composite_name*** file (also known as the **composite.xml** file) of the project in which to create a configuration plan, and select **Generate Config Plan**. Figure 47-1 provides details.

Figure 47-1 Generate a Configuration Plan



The Composite Configuration Plan Generator dialog appears, as shown in Figure 47-2.

Figure 47-2 Composite Configuration Plan Generator Dialog



3. Create a configuration plan file for editing, as shown in Table 47-2.

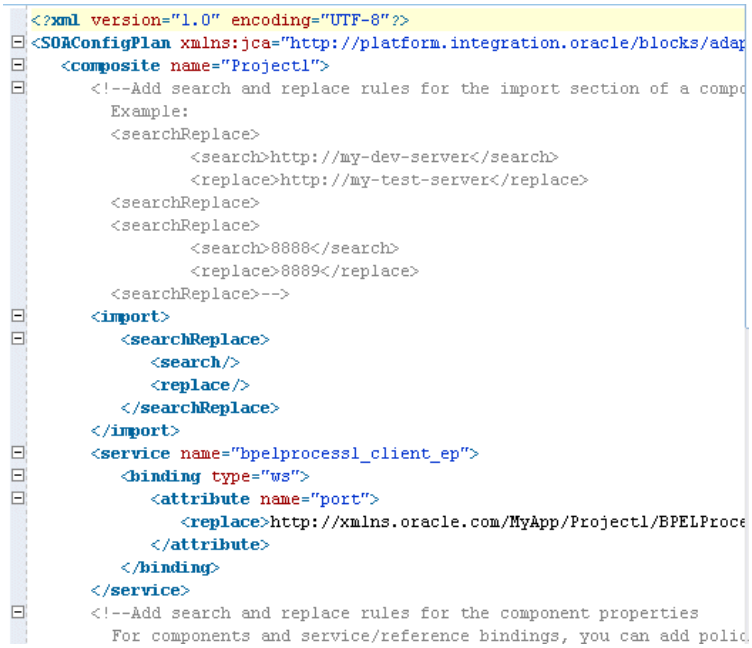
Table 47-2 Generate a Configuration Plan

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. For more information, see Deploying the Profile.
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. You can modify URLs and properties for the `composite.xml`, WSDL, and schema files of the SOA composite application. Figure 47-3 provides details.

Figure 47-3 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.

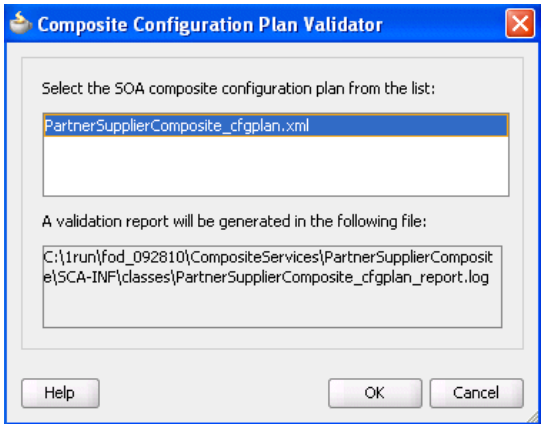
6. From the **File** menu, select **Save All**.

7. Above the editor, click the **x** to the right of the file name to close the configuration plan file.

8. In the Applications window, right-click the ***composite_name*** file again, and select **Validate Config Plan**.

   The Composite Configuration Plan Validator appears, as shown in Figure 47-4.

Figure 47-4 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:

    - How to Deploy a Single SOA Composite in
    - How to Deploy Multiple SOA Composite Applications in
    - How to Deploy and Use Shared Data Across Multiple SOA Composite Applications in

    During deployment in Oracle JDeveloper, the Deploy Configuration page shown in Step 4 of 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**.

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 about using these commands, see *WLST Command Reference for SOA Suite*.

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 How to Use ant to Deploy a SOA Composite Application.

How to Create Global Token Variables

You can define global token variables for specific URIs in SOA composite applications. For example, instead of updating the SOA composite application name in ten different configuration plans, you can set the name globally. The value is retrieved and replaces the value of the global token variable for the composite name in the `composite.xml` file of the deployed SOA composite application.

For more information, see *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

Deploying SOA Composite Applications in Oracle JDeveloper

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 data 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
- Deploying SOA composite applications with no server running

How to Deploy 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.

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 using the **IntegratedWebLogicServer** connection available with the Oracle SOA Suite Quick Start installation, see *Installing SOA Suite and Business Process Management Suite Quick Start for Developers*.

Note:

You can also create an application server connection by selecting **Window** > **Application Servers**, then right-clicking the **Application Servers** node in the Applications window and selecting **New Application Server**. This option prompts you to create a standalone server connection or Integrated WebLogic Server connection.

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 12.x** 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**.

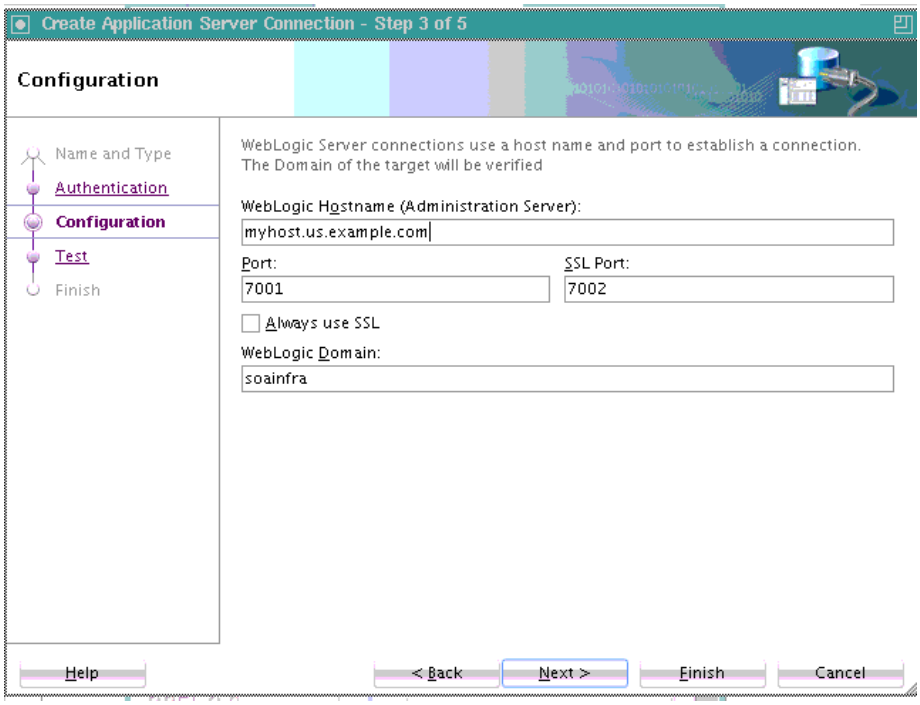
   The Configuration page appears.
10. In the **Weblogic Hostname (Administration Server)** field, enter the host on which the Oracle WebLogic Server is installed.
11. In the **Port** and **SSL Port** fields, enter appropriate port values or accept the default values.
12. If you want to use secure socket layer (SSL), select the **Always use SSL** check box. Table 47-3 describes what occurs when you select this check box.

Table 47-3 Deployment to HTTPS and HTTP Servers

If This Check Box 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**. Figure 47-5 provides details.

Figure 47-5 Server Name and Domain Selection

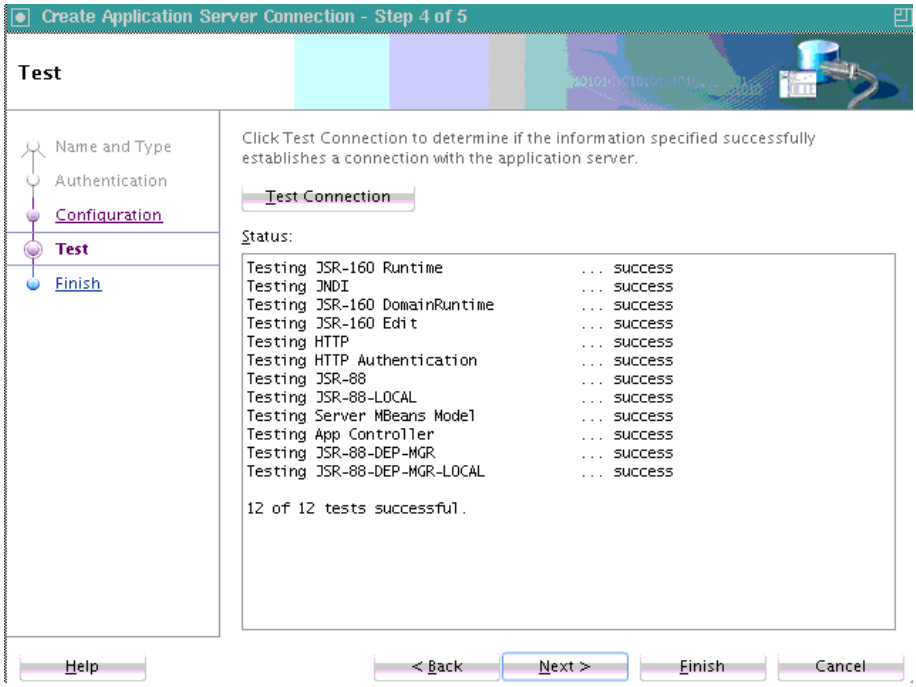


14. Click **Next**.

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. Figure 47-6 provides details.

Figure 47-6 Application Server Connection Test



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 Applications window, right-click the SOA project.
2. Select **Project Properties**.

   The Project Properties dialog appears.
3. Click **Deployment**.
4. Click the **New Profile** icon.

   The Create Deployment Profile dialog appears.
5. Enter the values shown in Table 47-4.

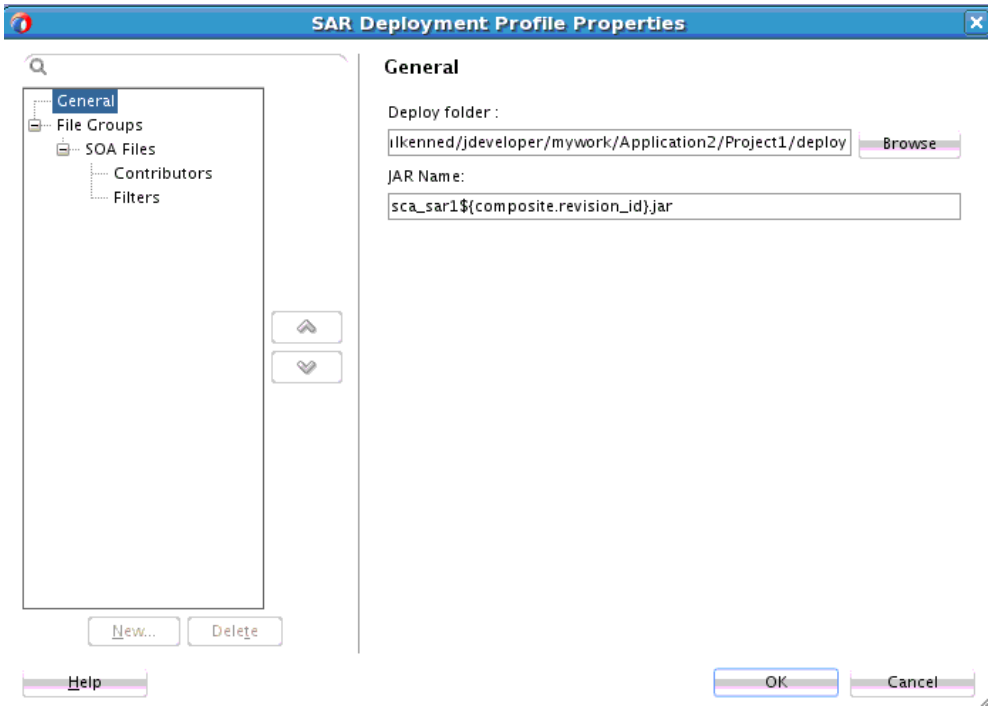
Table 47-4 Create Deployment Profile Dialog Fields and Values

Field	Description
Profile 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).
Deployment Profile Name	Enter a deployment profile name.
Description	Enter a description for the profile name.

6. Click **OK**.

   The SAR Deployment Profile Properties dialog appears, as shown in Figure 47-7.

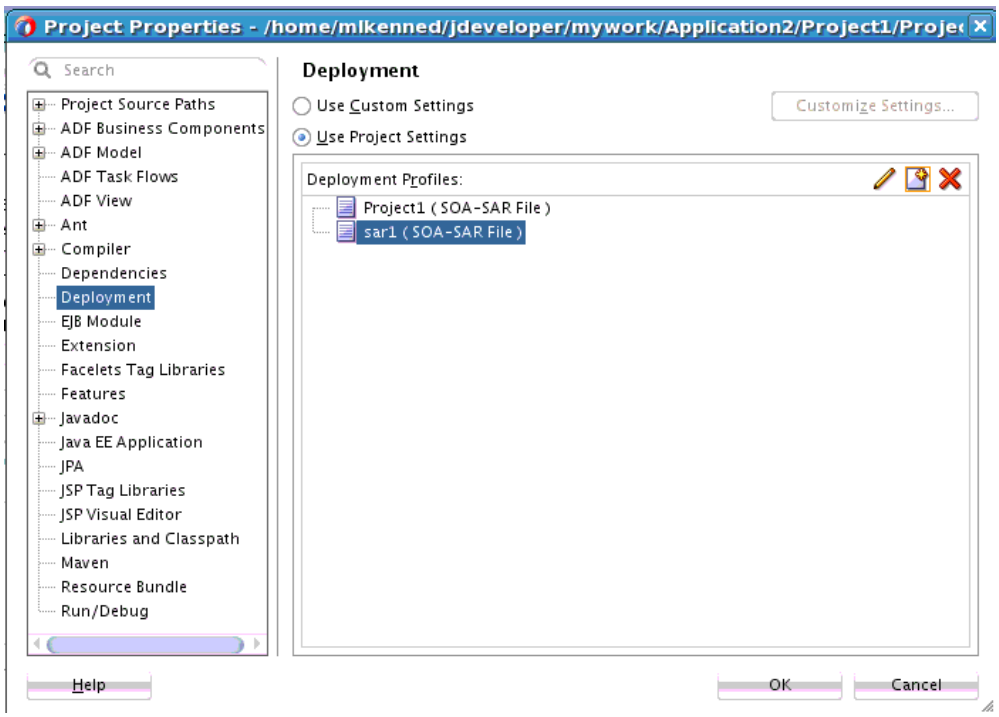
Figure 47-7 SAR Deployment Profile Properties



7. Optionally specify the target folder in which to save the SAR file.
8. Click **OK** to close the SAR Deployment Profile Properties dialog.

   The deployment profile shown in Figure 47-8 displays in the Project Properties dialog.

Figure 47-8 Project Profile



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 Creating an Application Server Connection.

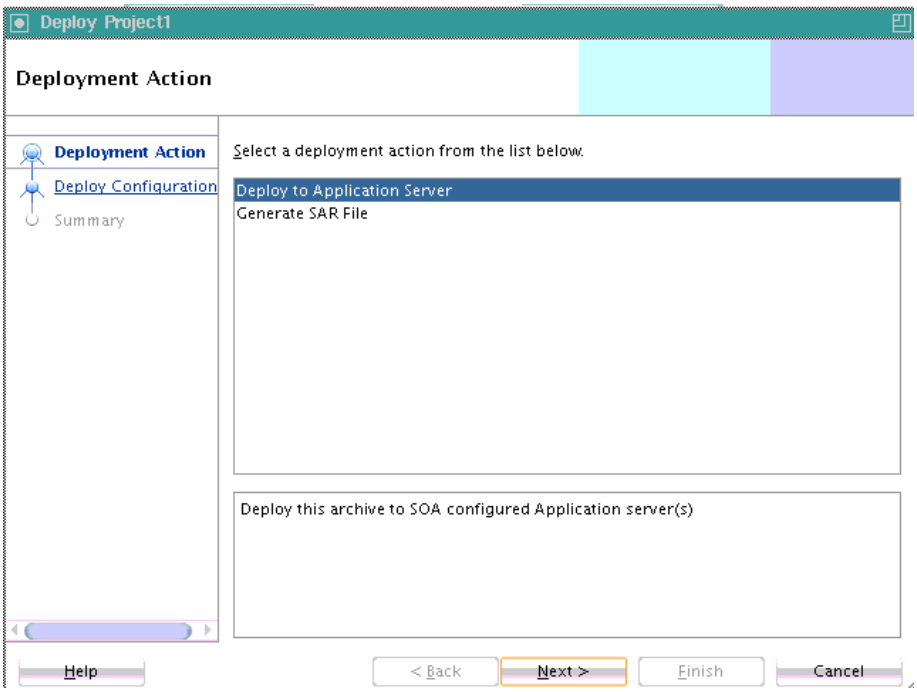
To deploy the profile:

1. In the Applications window, 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 47-9 provides an example.

Figure 47-9 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.

   - **Generate SAR File**

     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.

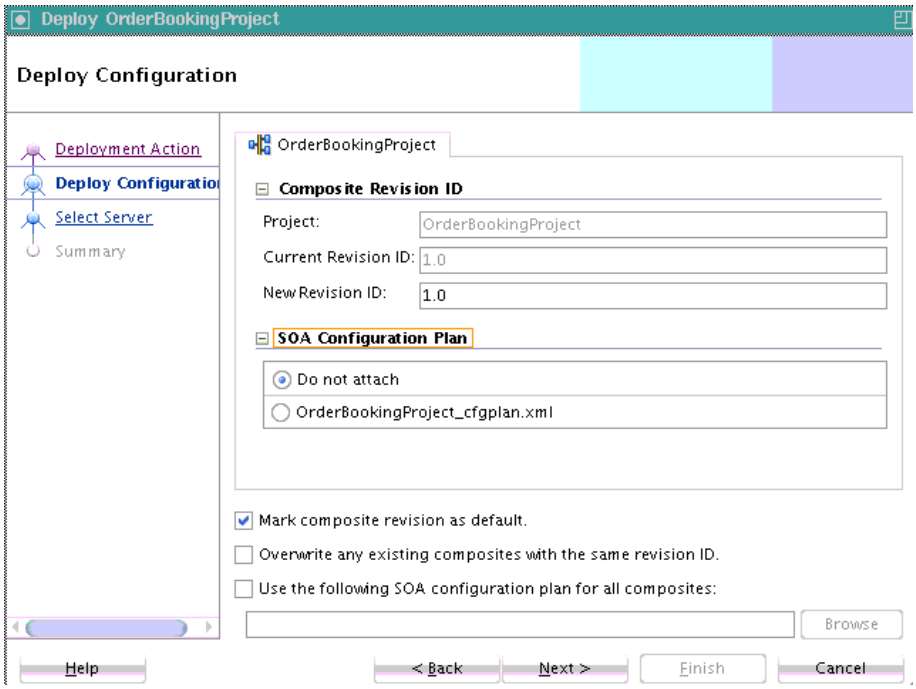
4. Select the deployment option appropriate for your environment. Table 47-5 provides details.

Table 47-5 Deployment Target

If You Select...	Go to...
Deploy to Application Server	Step 44.a
Generate SAR File	Step 44.b

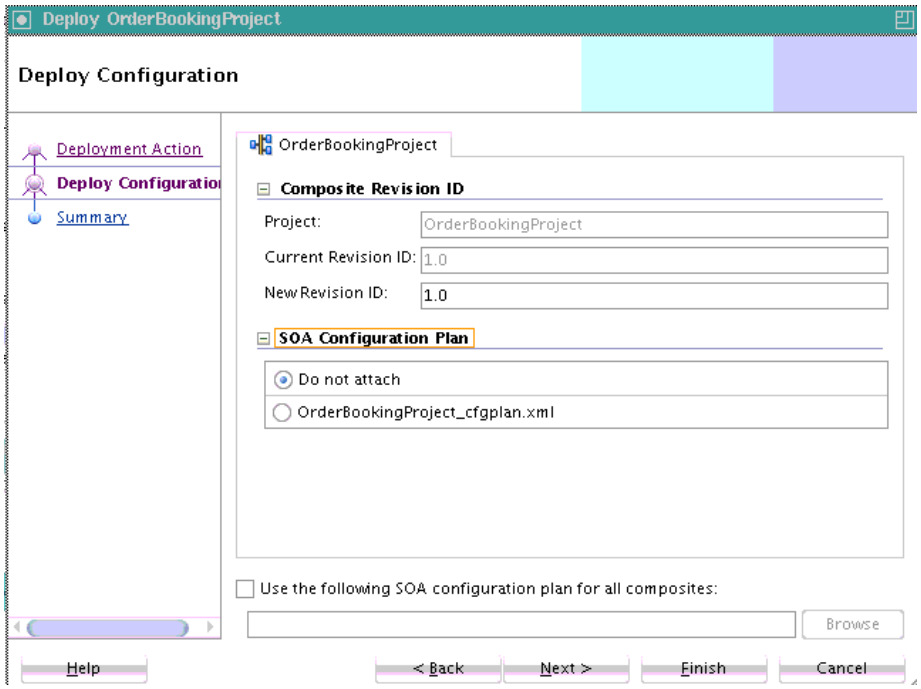
a. View the Deploy Configuration page shown in Figure 47-10.

Figure 47-10 Deploy Configuration Page for Application Server Deployment



b. View the Deploy Configuration page shown in Figure 47-11.

Figure 47-11 Deploy Configuration Page for Generate SAR File Deployment



5. Provide values appropriate to the deployment option you selected in Step 4, as described in Table 47-6. If you selected to deploy to an application server, additional fields are displayed.

Table 47-6 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. You can specify a new value or continue to use the current value. This revision ID becomes the value for the `${composite.revision_id}` variable in the application name. For example, if you enter `2.0` as the new revision ID for a composite named **OrderBooking**, `${composite.revision_id}` is replaced with `_rev2.0` (`sca_OrderBooking_rev2.0.jar`).
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.

Table 47-6 (Cont.) SOA Deployment Configuration Dialog

Field	Description
• **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 How to Use Configuration Plans to Customize SOA Composite Applications for the Target Environment for instructions on creating a configuration plan.
BPEL Monitor	Expand to display details about BPEL monitors.
• **Ignore BPEL Monitor deployment errors** **Note:** This check box only appears if there is at least one `.monitor` file in the application.	Deselect this check box to display BPEL Monitor deployment errors. This check box 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. This 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. This option only displays if you selected **Deploy to Application Server** on the Deployment Action page.

Table 47-6 (Cont.) SOA Deployment Configuration Dialog

Field	Description
Keep running instances on after redeployment	**Note**: This option is displayed if Oracle BPM Suite is installed in Oracle JDeveloper, and only supported for the deployment of Oracle BPM composites. Do *not* select this option if you are deploying: • A SOA composite application from an Oracle JDeveloper environment in which Oracle BPM Suite is also installed. • An Oracle BPM composite that includes a durable BPEL process, regardless of whether that process has been modified. Durable BPEL processes are those that take time to complete execution. Examples of durable BPEL processes are asynchronous processes (which are always durable) and synchronous processes that include a durable activity such as a wait activity. If you select this option and attempt to redeploy a durable BPEL process, then deployment fails. Select to enable existing instances of the overwritten revision to continue running instead of being aborted. These instances run side by side with any new instances that you create with the new revision of the Oracle BPM composite application.
Force deployment of incompatible processes	This option is only displayed for Oracle BPM Suite composites. If **Keep running instances on after redeployment** is checked, this option is displayed. Select this check box to force deployment of incompatible BPM processes. When a composite with BPM processes is overwritten, the system checks to see if the BPM processes being overwritten are compatible with the processes being deployed. If they are compatible, running instances of these processes are not marked as aborted and deployment is successful. If they are incompatible, deployment fails unless you select this check box.
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.

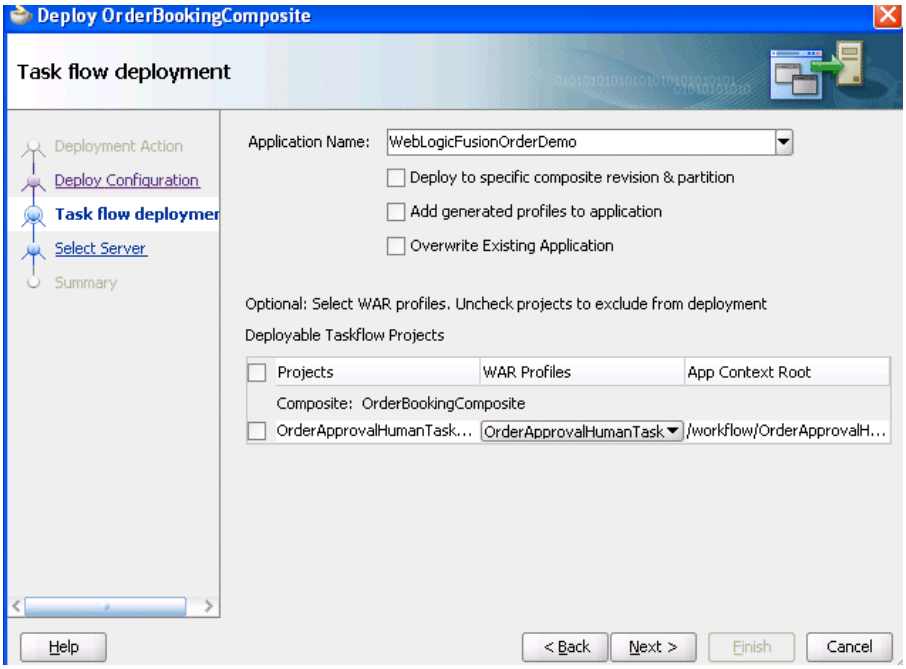
6. When finished, click **Next**.

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 47-12.

   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 Taskflow Projects** table of this dialog.

Figure 47-12 Task Flow Deployment Page



8. Provide values appropriate to your environment, as described in Table 47-7.

Table 47-7 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 an EAR profile to deploy based on the WAR profiles selected in the **Deployable Taskflow 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 check box 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.

Table 47-7 (Cont.) Task Flow Deployment Dialog

Field	Description
Deployable Taskflow 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** check box is selected. **Note**: If you do not select a WAR profile, no task flows are deployed.
• **Projects**	Select from the list of deployable task flow projects or select the **Projects** check box 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.

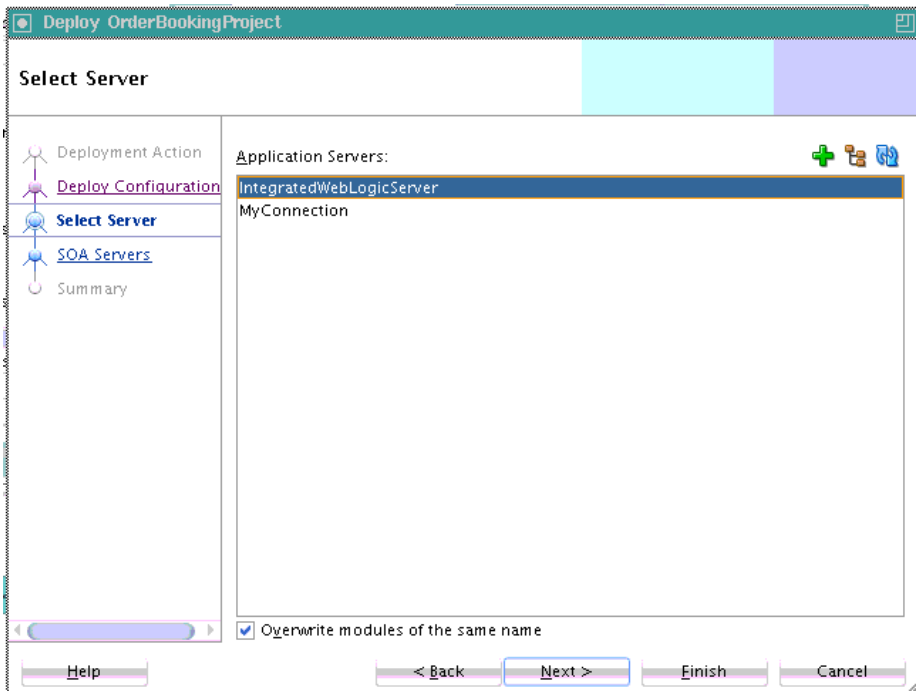
For information about deploying SOA composite applications with task flows to multiple partition environments, see What You May Need to Know About Deploying Human Task Composites with Task Flows to Partitions.

9. Click **Next**.

10. If you selected to deploy to an application server in Step 3, 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 47-13 provides details.

    If you selected to generate a SAR file in Step 3, go to Step 15.

Figure 47-13 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 47-14 provides details.

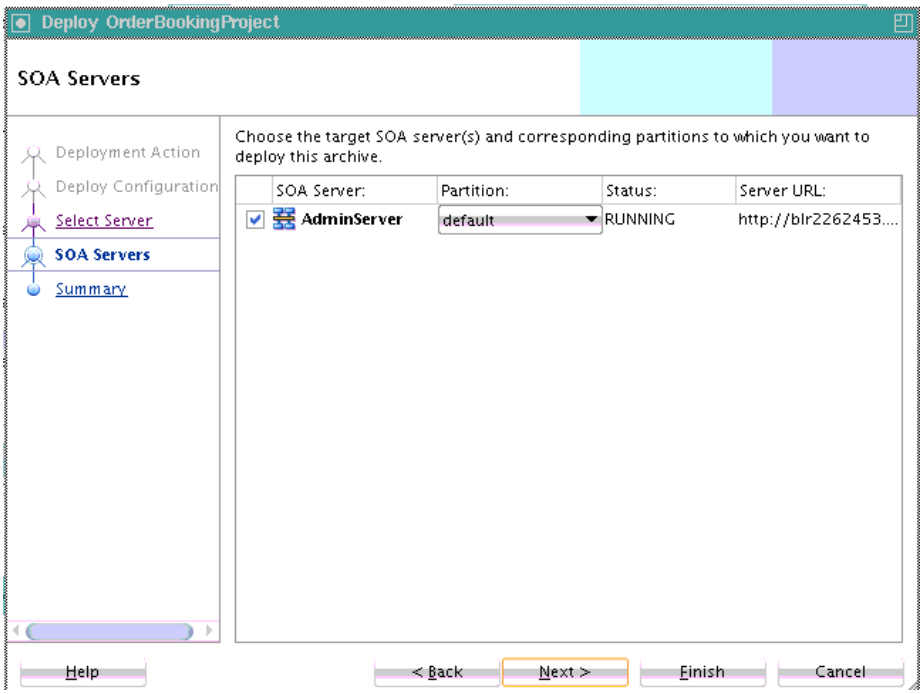
13. Select the partition in which to deploy this archive. If the server contains no partitions, you cannot deploy this archive. Also, if the server is not in a *running* state, you cannot deploy this archive. By default, a partition named **default** is automatically included with Oracle SOA Suite. You 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.

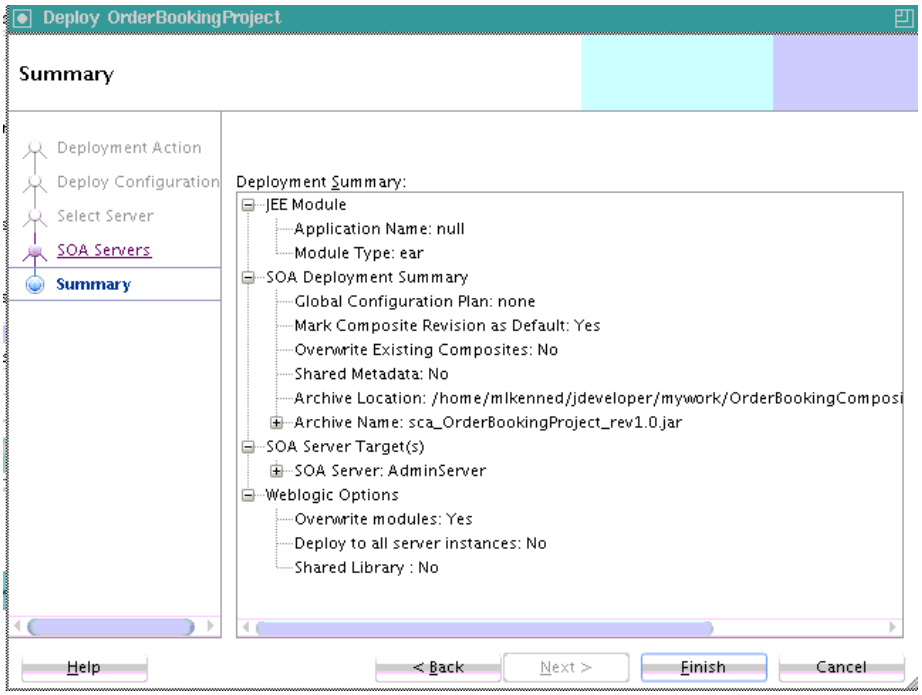
Figure 47-14 SOA Servers Page



14. Click **Next**.

15. Review the archive details on the Summary page shown in Figure 47-15, and click **Finish**.

Figure 47-15 Summary Page



16. If you selected to deploy to an application server in Step 3, 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 with a naming convention of **sca_*composite_name*_rev*revision_number*.jar**.
    - The project is displayed in the Resources window under ***application_server_connection_name*** > **SOA** > ***SOA_server_name*** > ***partition_name***.
    - The project is displayed in the Application Servers window 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 *Administering Oracle SOA Suite and Oracle Business Process Management 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 Testing and Troubleshooting.

    For information about creating partitions, see the following documentation:

    - Deploying and Managing SOA Composite Applications with ant Scripts
    - *Administering Oracle SOA Suite and Oracle Business Process Management Suite*
    - *WLST Command Reference for SOA Suite*

    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** check box on the Deploy Configuration page.

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`). The following example provides details:

```
<hwTaskFlows
 xmlns="http://xmlns.oracle.com/bpel/workflow/hwTaskFlowProperties">
   <ApplicationName>worklist</ApplicationName>
   <LookupType>LOCAL</LookupType>
   <TaskFlowDeploy>false</TaskFlowDeploy>
   <PartitionName>partition2</PartitionName>
```

If you want to deploy the task flow for the SOA composite application on all partitions, leave **PartitionName** blank. If you want to use different task flows for the composites on different partitions, then **PartitionName** must be specified.

In addition, if you want to reuse the same task flow project for another partition, you must change the web context root.

How to Deploy 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.

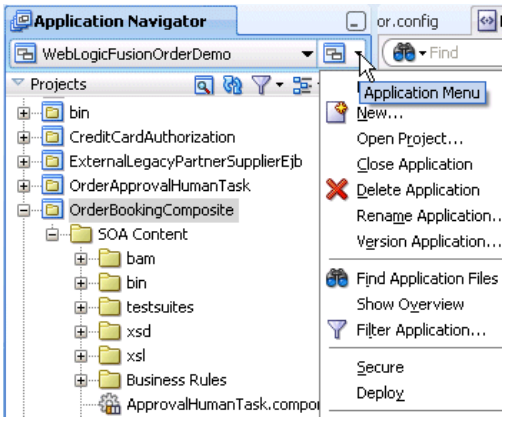
Note:

- This section assumes you have created an application server connection. If not, see Creating an Application Server Connection for instructions.
- 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.

To deploy multiple SOA composite applications:

1. From the **Application** menu, select **Application Properties**, as shown in Figure 47-16.

Figure 47-16 Application Properties

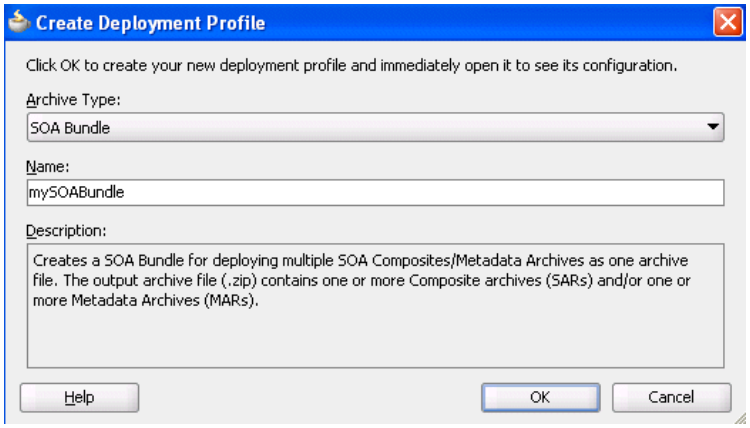


2. In the **Application Properties** dialog, click **Deployment**.
3. Click **New**.

   The Create Deployment Profile dialog appears.
4. In the **Archive Type** list, select **SOA Bundle**.
5. In the **Name** field, enter a name.

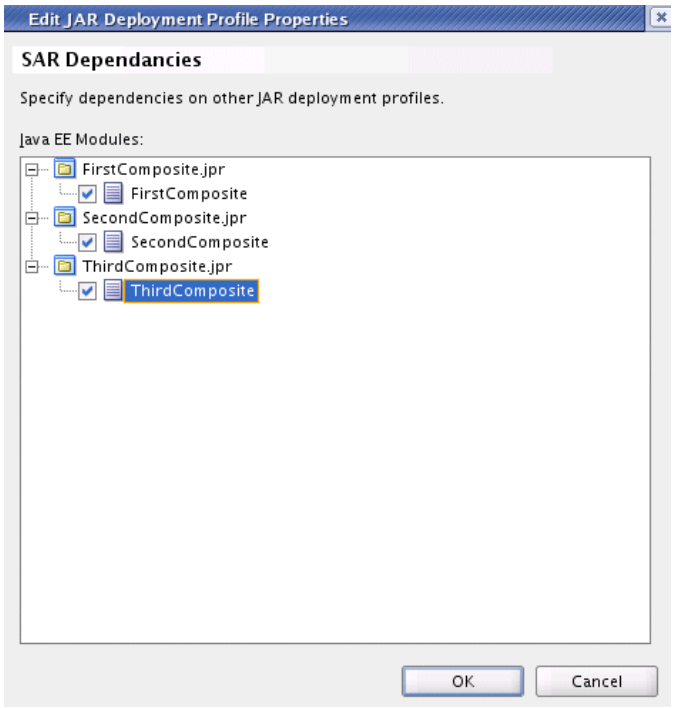
   Figure 47-17 provides details.

Figure 47-17 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 47-18.

Figure 47-18 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 for details about responses to provide.

How to Deploy and Use Shared Data Across Multiple SOA Composite Applications in Oracle JDeveloper

This section describes how to deploy and use shared data such as WSDLs, XSDs, and other file types across multiple SOA composite applications.

Shared data 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 data is deployed to an existing SOA Infrastructure partition on the server. This data 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`.

Note:

- You always deploy to the `/apps` location. The directory hierarchy must exist in the JAR file to deploy. Do not create the directory hierarchy in the Oracle MDS Repository first and then deploy the JAR file to that location. For example, to deploy to `/apps/demo/credit card`, the JAR file must include the `demo/credit card` directory hierarchy inside it.
- Files that begin with a period (for example, `.designer`) cannot be shared across SOA composite applications.

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

Create a JAR Profile and Include the Artifacts to Share

To create a JAR profile and include the artifacts to share:

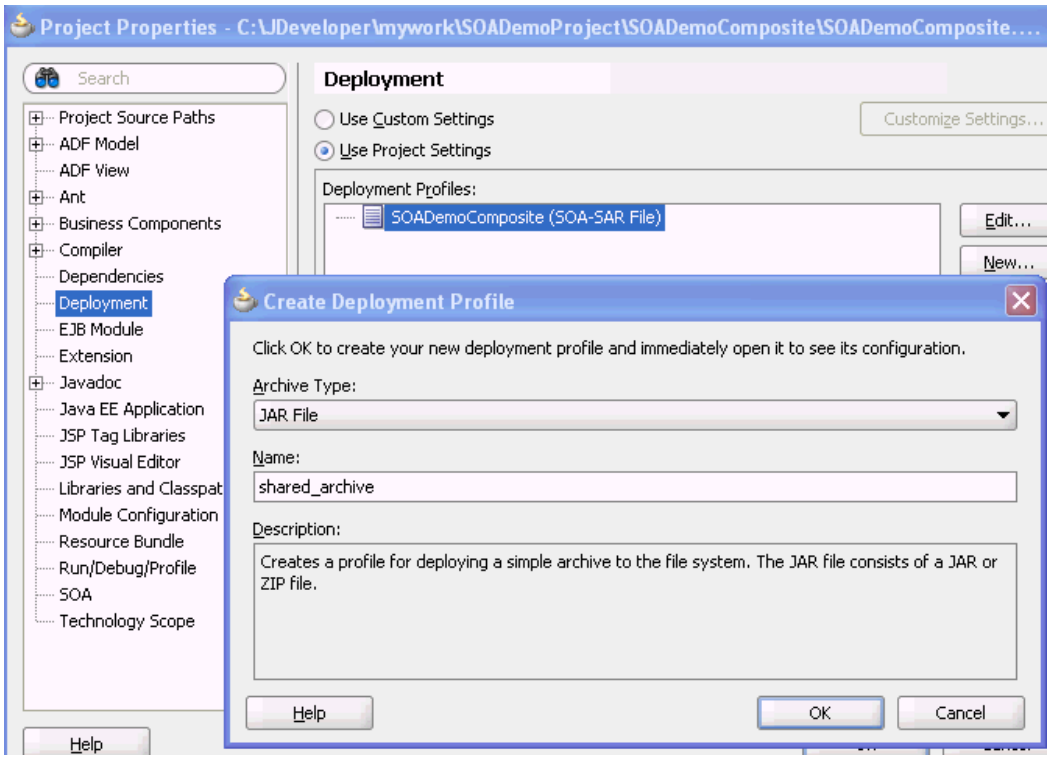
1. In the Applications window, right-click the SOA project.
2. Select **Project Properties**.

   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 47-19.

Figure 47-19 JAR File Selection



7. Click **OK**.

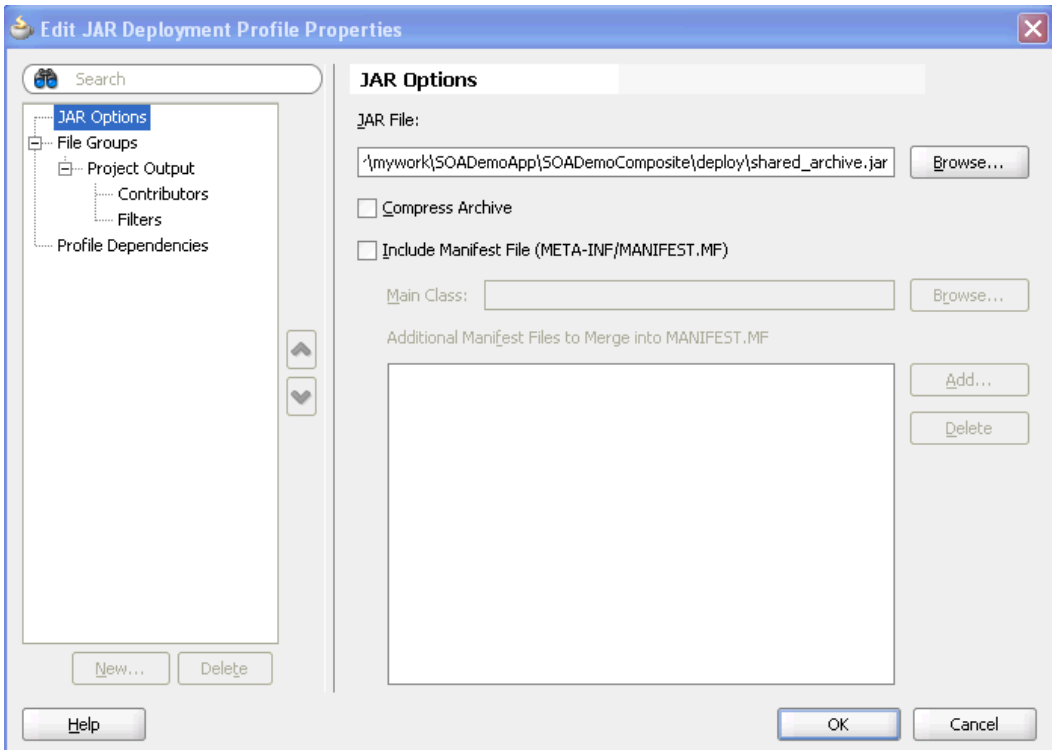
   The JAR Deployment Profile Properties dialog appears.

8. Select **JAR Options** from the navigational tree on the left.

9. Deselect **Include Manifest File (META-INF/MANIFEST.MF)**, as shown in Figure 47-20.

   This prevents the archive generator from adding the manifest file (`META-INF/MANIFEST.MF`) into the JAR file.

Figure 47-20 JAR File Options

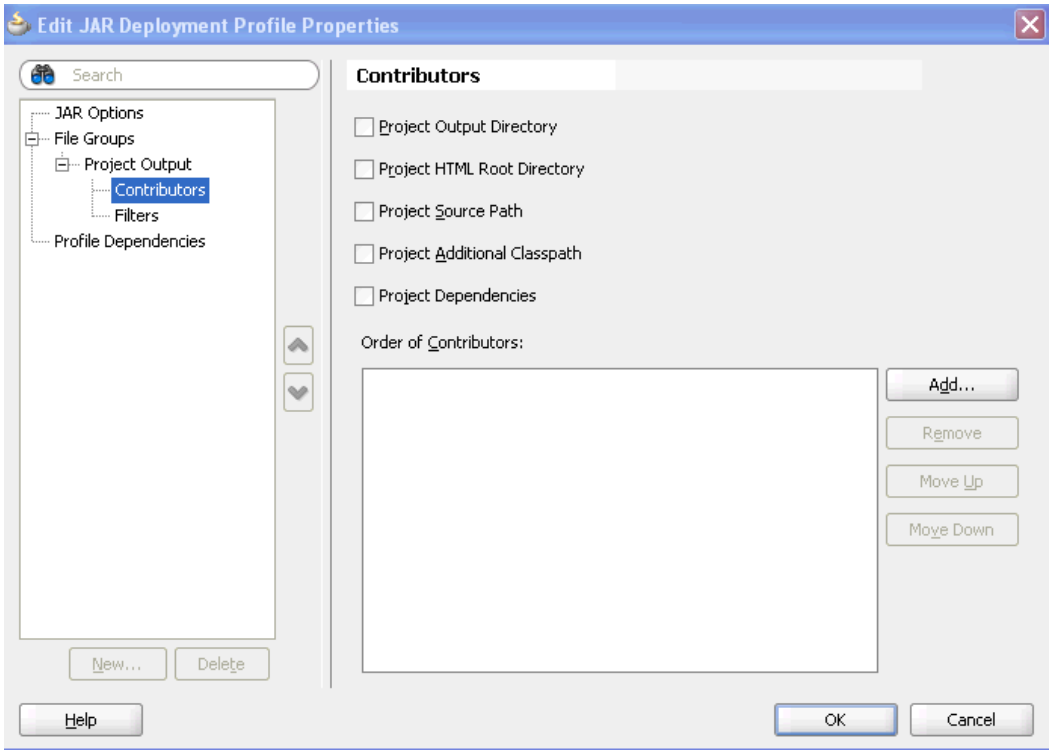


10. Select **File Groups** > **Project Output** > **Contributors** from the navigational tree on the left.

11. Deselect the **Project Output Directory** and **Project Dependencies** options, as shown in Figure 47-21.

    This prevents the archive generator from adding the contents of the project output and project dependencies into the archive.

Figure 47-21 Contributors



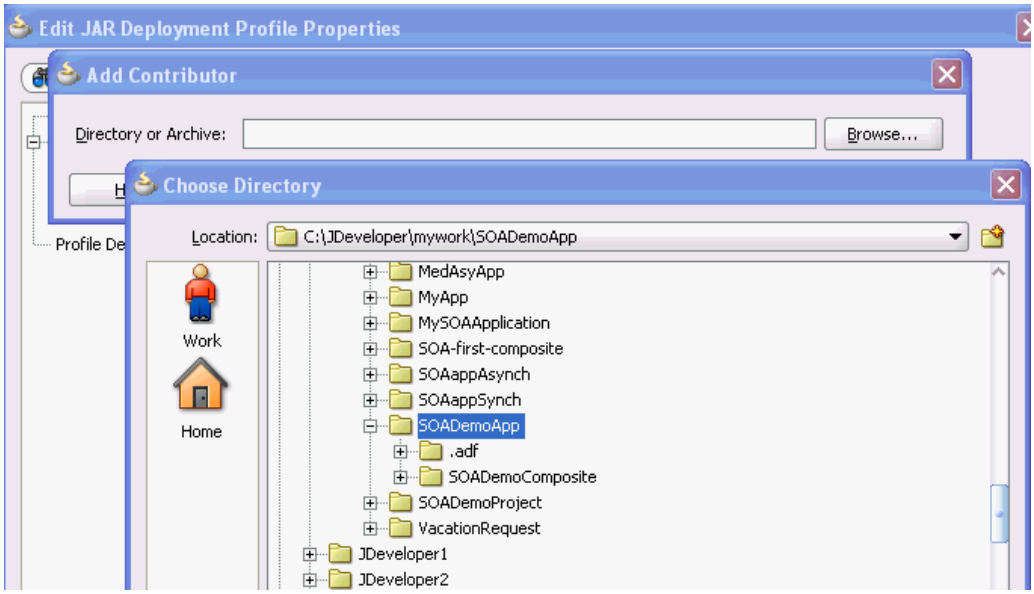
12. Click **Add** to add a new contributor.

    The Add Contributor dialog appears. This dialog enables you to add artifacts to your archive.

13. Click **Browse**.

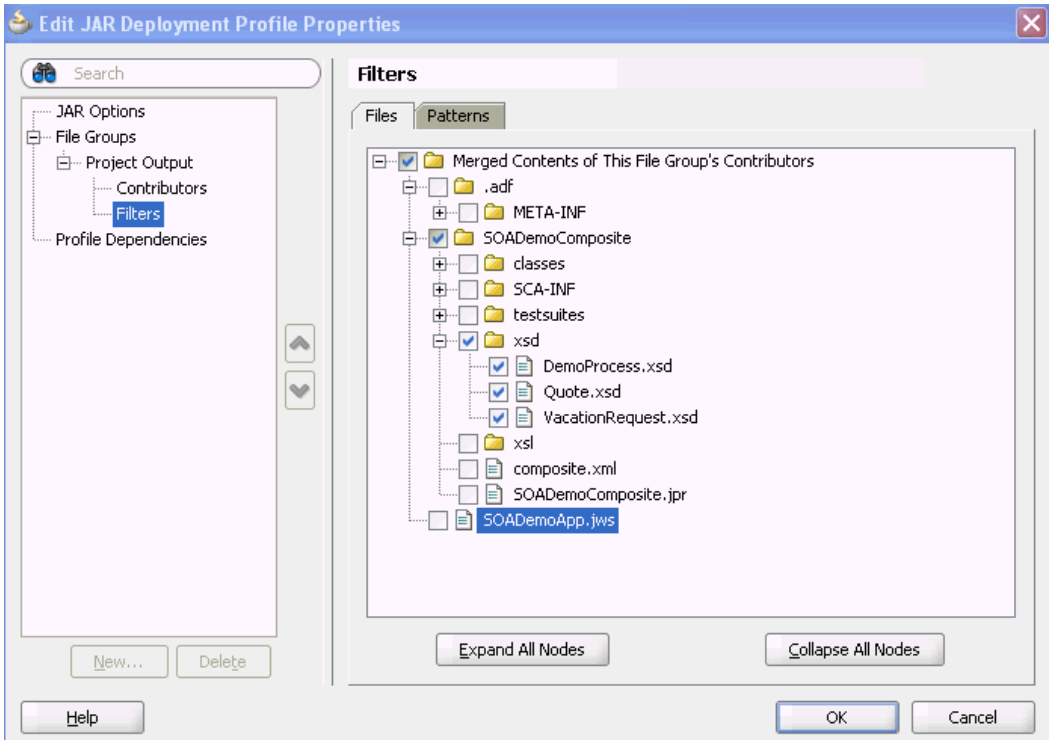
14. Select the folder in which your artifacts reside, as shown in Figure 47-22. This selection also determines the hierarchy of artifacts in the archive.

Figure 47-22 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 47-23. For this example, the archive contains the following XSD files:
    - **SOADemoComposite/xsd/DemoProcess.xsd**
    - **SOADemoComposite/xsd/Quote.xsd**
    - **SOADemoComposite/xsd/VacationRequest.xsd**

Figure 47-23 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**.

Create a SOA Bundle that Includes the JAR Profile

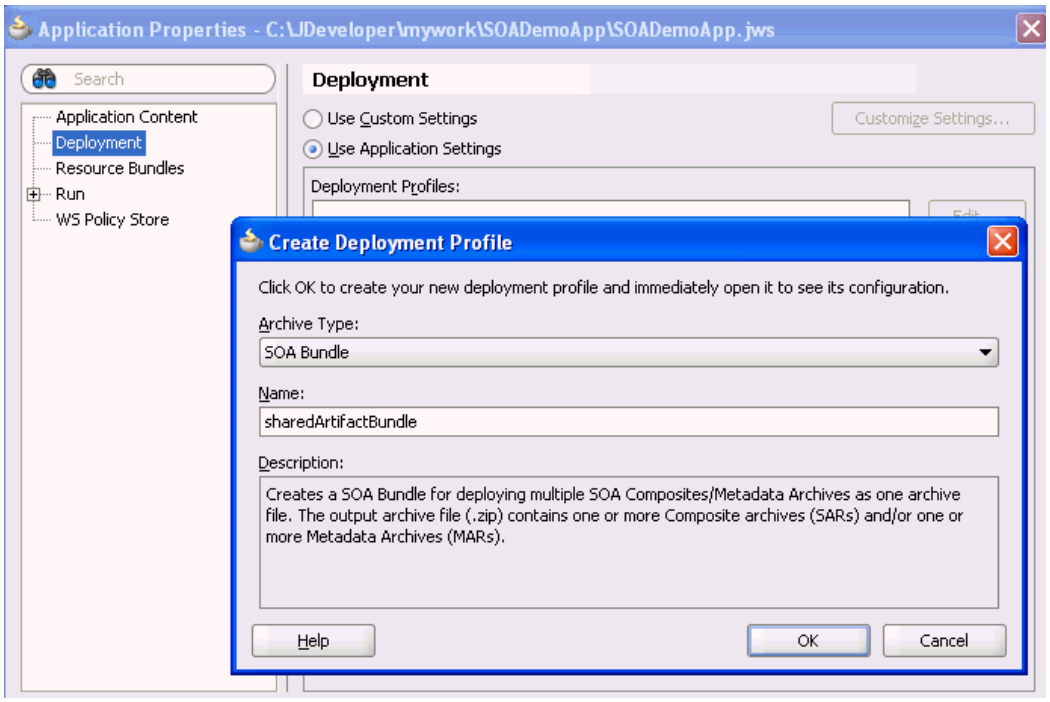
To create a SOA bundle that includes the JAR profile:

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 47-24 provides details.

Figure 47-24 SOA Bundle Creation

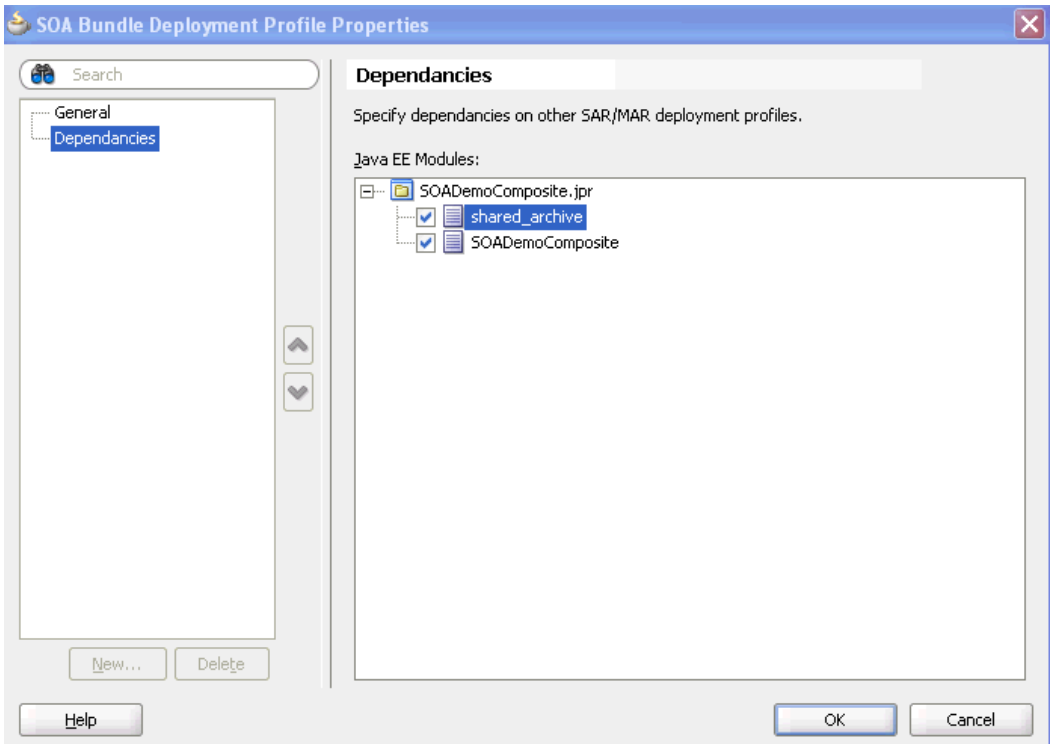


5. Click **OK**.

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 47-25 provides details.

Figure 47-25 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**.

Deploy the SOA Bundle with Oracle JDeveloper

To deploy the SOA bundle with Oracle JDeveloper:

1. Right-click the **Application** menu and select **Deploy** > *SOA_Bundle_Name*.

   This invokes the deployment wizard.

2. See Step 3 of Deploying the Profile for details about responses to provide.

   This deploys the SOA bundle to the application server (shared artifacts are deployed to the Oracle MDS Repository database of Oracle SOA Suite).

To deploy the SOA bundle with ant:

See How to Use ant to Deploy a SOA Composite Application.

Use Shared Data

This section describes how to browse and select the shared data you created in How to Deploy and Use Shared Data Across Multiple SOA Composite Applications in .

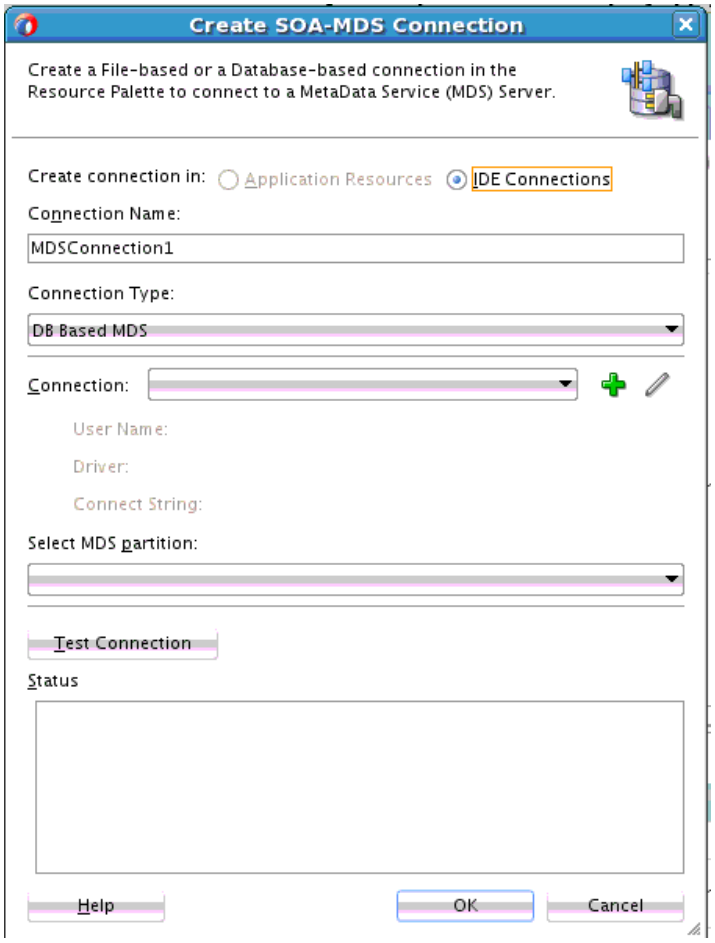
Creating a SOA-MDS Connection

To create a SOA-MDS connection:

1. From the **File** menu, select **New** > **Application** > **Connections** > **SOA-MDS Connection**.

   The Create SOA-MDS Connection dialog shown in Figure 47-26 is displayed.

Figure 47-26 Create SOA-MDS Connection Dialog



2. Provide values appropriate to your environment, as shown in Table 47-8.

Table 47-8 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 Resources window 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 Resources window.

Table 47-8 (Cont.) Create SOA-MDS Connection Dialog

Field	Description
Connection Type	Select a connection type. An Oracle 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.
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 Resources window and view shared artifacts under the **/apps** node.

Creating a BPEL Process

You can now browse and use the shared data 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 **SOA-MDS**.
6. Select shared data. For this example, the **Quote.xsd** file that you selected to include in the archive in Step 18 of "Create a JAR Profile and Include the Artifacts to Share is selected.

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 Applications window, 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.

    The Type Chooser dialog includes a **Recent Files** folder in which information is kept for the duration of the Oracle JDeveloper session. For example, if you create a new BPEL process and want to define the input variable from a schema in the SOA Design-Time MDS Repository, you go there once. When you want to define the output variable from the same schema, the schema remains visible in the **Recent Files** folder.

How to Deploy an Existing SOA Archive in Oracle JDeveloper

You can deploy an existing SOA archive from the Application Servers window in Oracle JDeveloper.

Note:

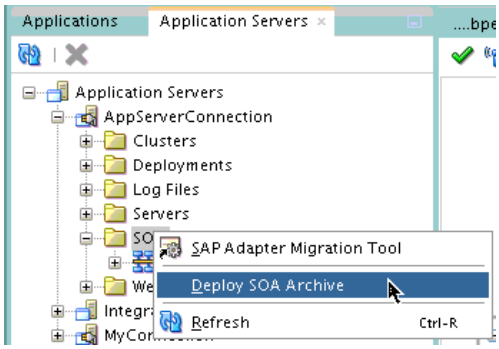
- The archive must exist. You cannot create an archive in the Deploy SOA Archive dialog.
- These instructions assume you have created an application server connection to an Oracle WebLogic Administration Server or another supported application server on which the SOA Infrastructure is deployed. Creating 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** > **Application** > **Connections** > **Application Server Connection** to create a connection.

To deploy an existing SOA archive from Oracle JDeveloper:

1. From the **Window** menu, select **Application Servers**.
2. In the Applications window, expand your connection name.

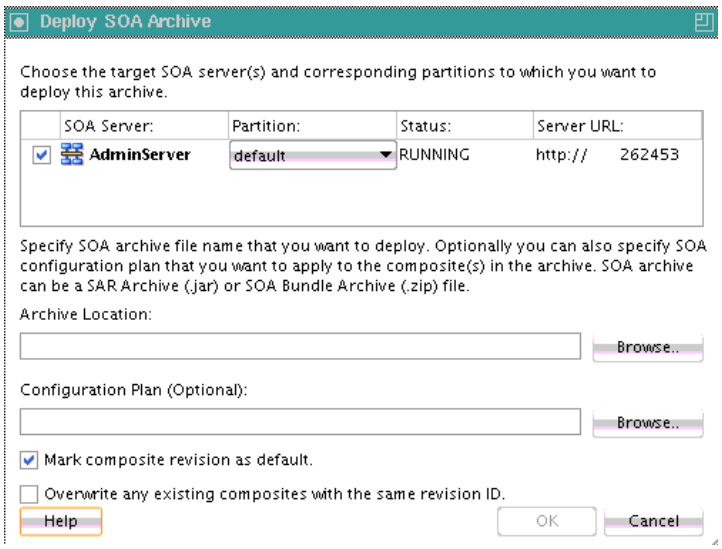
3. Right-click the **SOA** folder.
4. Select **Deploy SOA Archive**.

Figure 47-27 SOA Archive Deployment from the Applications Window



The Deploy SOA Archive dialog shown in Figure 47-28 appears.

Figure 47-28 Deploy SOA Archive Dialog



5. Provide responses appropriate to your environment, as described in Table 47-9.

Table 47-9 Deploy SOA Archive 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.

Table 47-9 (Cont.) Deploy SOA Archive Dialog Fields and Values

Field	Description
Archive Location	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 How to Create a Configuration Plan in or 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 aborted.

6. Click **OK**.

For more information on deploying and testing SOA composite applications from the Application Servers window, see Managing and Testing a SOA Composite Application.

Deploying and Managing SOA Composite Applications with the WLST Utility

You can manage SOA composite applications with the WLST utility. This utility is well-suited for automation and can be easily integrated into existing release processes. For instructions, see *WLST Command Reference for SOA Suite*.

Deploying and Managing 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. The `ant` utility is well-suited for automation and can be easily integrated into existing release processes.

Note:

Before using the Oracle SOA Suite `ant` scripts, you must first run the `setDomainEnv.sh` script (for Linux) or `setDomainEnv.cmd` script (for Windows). This script adds the necessary JAR files for using `ant` to the classpath.

Table 47-10 lists the `ant` scripts available in the *`Middleware_Home`*`\`*`SOA_Suite_Home`*`\bin` directory.

Table 47-10 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.

Table 47-10 (Cont.) ant Management Scripts

Script	Description
`ant-sca-mgmt.xml listDeployedComposites`	Lists deployed SOA composite applications.
`ant-sca-mgmt.xml listPartitions`	Lists all available partitions in the SOA Infrastructure.
`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 Oracle Enterprise Service Bus (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`

How to Use ant to Automate the Testing of a SOA Composite Application

The following provides an example of executing a test case. Test cases enable you to automate the testing of SOA composite applications:

```
ant -f ant-sca-test.xml -Dscatest.input=MyComposite
-Djndi.properties=/home/jdoe/jndi.properties
```

Table 47-11 describes the syntax.

Table 47-11 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.example.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.example.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 Automating Testing of SOA Composite Applications and *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

How to Use ant to Compile a SOA Composite Application

The following provides an example of compiling a SOA composite application, which validates it for structure and syntax:

```
ant -f ant-sca-compile.xml
-Dsca.input=/myApplication/myComposite/composite.xml
```

Table 47-12 describes the syntax.

Table 47-12 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. • `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 Oracle JDeveloper application home directory of the SOA composite application being compiled that contains the `.adf` directory in it. • `scac.displayLevel` Controls the level of logs written to `scac.output` file. The value can be `1`, `2`, or `3` (this defaults to `1`).

How to Use ant to Package a SOA Composite Application into a Composite SAR File

The following 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.

```
ant -f ant-sca-package.xml
-DcompositeDir=C:\demo\end2end-105-POProcessing\po\solutions\ch9\POProcessing\POPr
ocessing
-DcompositeName=POProcessing
-Drevision=6-cmdline
-Dsca.application.home=C:\demo\end2end-105-POProcessing\po\solutions\ch9\POProces
sing
```

Table 47-13 describes the syntax.

Table 47-13 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.

Table 47-13 (Cont.) ant Packaging Commands

Argument	Definition
`sca.application.home`	Absolute path of the application home directory. This property is required if your SOA composite application accesses shared artifacts in the MDS Repository. If not, it is optional.
`oracle.home`	Optional. The `oracle.home` property.

How to Use ant to Deploy a SOA Composite Application

The following provides an example of deploying a SOA composite application. You can also use this command to deploy shared data such as WSDLs, XSDs, and other file types across SOA composite applications. For information about shared data, see How to Deploy and Use Shared Data Across Multiple SOA Composite Applications in .

```
ant -f ant-sca-deploy.xml
-DserverURL=http://localhost:8001
-DsarLocation=C:\demo\end2end-105-POProcessing\po\solutions\ch9\POProcessing\POProcessing\deploy\sca_POProcessing_rev6-cmdline.jar
-Doverwrite=true
-Duser=weblogic
-DforceDefault=true
-Dconfigplan=C:\demo\end2end-105-POProcessing\po\solutions\ch9\POProcessing\POProcessing\demed_cfgplan.xml
-Dscac.user.classpath=C:\jarfolder\custom.jar
-Dpartition=partition.name
```

Note:

After specifying the user name, enter the password when prompted.

Table 47-14 describes the syntax.

Table 47-14 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.

Table 47-14 (Cont.) ant Deployment Commands

Argument	Definition
`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`
`scac.user.classpath`	Optional. The name of the external custom library. If you have a SOA composite application with a BPEL process service component that refers to a custom JAR file, set this property.
`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.

How to Use ant to Undeploy a SOA Composite Application

The following provides an example of 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 47-15 describes the syntax.

Table 47-15 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.
`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.

How to Use ant to Export a Composite into a SAR File

The following provides an example of 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 47-16 describes the syntax.

Table 47-16 ant Export Commands

Argument	Definition
`serverURL`	The URL of the server that hosts the SOA Infrastructure application (for example, `http://myhost:8001`).

Table 47-16 (Cont.) ant Export Commands

Argument	Definition
`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.

The following example shows how to export a composite without including any postdeployment changes:

```
ant -f ant-sca-deploy.xml exportComposite -DserverURL=http://myhost:8001
 -DupdateType=none
 -DsarFile=/tmp/sca_HelloWorld_rev1.0.jar -DcompositeName=HelloWorld
 -Drevision=1.0
```

The following example shows how to export a composite with all postdeployment changes:

```
ant -f ant-sca-deploy.xml exportComposite -DserverURL=http://myhost:8001
 -DupdateType=all
 -DsarFile=/tmp/sca_HelloWorld_rev1.0-all.jar -DcompositeName=HelloWorld
 -Drevision=1.0
```

The following example shows how to export a composite with property postdeployment updates:

```
ant -f ant-sca-deploy.xml exportComposite -DserverURL=http://myhost:8001
 -DupdateType=property
 -DsarFile=/tmp/sca_HelloWorld_rev1.0-prop.jar -DcompositeName=HelloWorld
 -Drevision=1.0
```

The following example shows how to export a composite with runtime/metadata postdeployment updates:

```
ant -f ant-sca-deploy.xml exportComposite -DserverURL=http://myhost:8001
 -DupdateType=runtime
 -DsarFile=/tmp/sca_HelloWorld_rev1.0-runtime.jar
 -DcompositeName=HelloWorld -Drevision=1.0
```

How to Use ant to Export Postdeployment Changes of a Composite into a JAR File

The following provides an example of 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 47-17 describes the syntax.

Table 47-17 ant Postdeployment Export Commands

Argument	Definition
`serverURL`	The URL of the server that hosts the SOA Infrastructure application (for example, `http://myhost: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)).
`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.

The following example shows how to export all postdeployment updates:

```
ant -f ant-sca-deploy.xml exportUpdates -DserverURL=http://myhost:8001
 -DupdateType=all
 -DjarFile=/tmp/all-HelloWorld_rev1.0.jar -DcompositeName=HelloWorld
 -Drevision=1.0
```

The following example shows how to export property postdeployment updates:

```
ant -f ant-sca-deploy.xml exportUpdates -DserverURL=http://myhost:8001
 -DupdateType=property
 -DjarFile=/tmp/prop-HelloWorld_rev1.0.jar -DcompositeName=HelloWorld
 -Drevision=1.0
```

The following example shows how to export runtime/metadata postdeployment updates.

```
ant -f ant-sca-deploy.xml exportUpdates -DserverURL=http://myhost:8001
 -DupdateType=runtime
 -DjarFile=/tmp/runtime-HelloWorld_rev1.0.jar -DcompositeName=HelloWorld
 -Drevision=1.0
```

How to Use ant to Import Postdeployment Changes of a Composite

The following provides an example of 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 47-18 describes the syntax.

Table 47-18 ant Postdeployment Import Commands

Argument	Definition
`serverURL`	The URL of the server that hosts the SOA Infrastructure application (for example, `http://myhost: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.

The following example shows how to import postdeployment changes of a composite:

```
ant -f ant-sca-deploy.xml importUpdates -DserverURL=http://myhost:8001
 -DjarFile=/tmp/prop-HelloWorld_rev1.0.jar -DcompositeName=HelloWorld
 -Drevision=1.0
```

How to Use ant to Export Shared Data of a Given Pattern into a JAR File

The following provides an example of 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 47-19 describes the syntax.

Table 47-19 ant Shared Data Export Commands

Argument	Definition
`serverURL`	The URL of the server that hosts the SOA Infrastructure application (for example, `http://myhost:8001`).
`jarFile`	The absolute path of the JAR file to be generated.
`pattern`	The file pattern supported by Oracle MDS Repository 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.

The following example shows how to export shared data of a given pattern into a JAR file.

```
ant -f ant-sca-deploy.xml exportSharedData -DserverURL=http://myhost:8001
 -DjarFile=/tmp/MySharedData.jar
 -Dpattern="/Project1/**"
```

How to Use ant to Remove a Top-level Shared Data Folder

The following provides an example of removing a top-level shared data folder, even if there are composites deployed in the service engine:

```
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 47-20 describes the syntax.

Table 47-20 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.

The following example shows how to remove a top-level shared data folder named `Project1`:

```
ant -f ant-sca-deploy.xml removeSharedData -DserverURL=http://myhost:8001
 -DfolderName=Project1
```

How to Use ant to Start a SOA Composite Application

The following provides an example of starting a SOA composite 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 47-21 describes the syntax.

Table 47-21 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.

Table 47-21 (Cont.) ant SOA Composite Application Startup 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.

How to Use ant to Stop a SOA Composite Application

The following provides an example of stopping a SOA composite 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 47-22 describes the syntax.

Table 47-22 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.

How to Use ant to Activate a SOA Composite Application

The following provides an example of activating a SOA composite 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 47-23 describes the syntax.

Table 47-23 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.

How to Use ant to Retire a SOA Composite Application

The following provides an example of retiring a SOA composite 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 47-24 describes the syntax.

Table 47-24 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`).
`user`	User name for connecting to the running server to get MBean information (for example, `weblogic`).
`password`	Password for the user name.

Table 47-24 (Cont.) ant SOA Composite Application Retirement Commands

Argument	Definition
`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.

How to Use ant to Assign the Default Version to a SOA Composite Application

The following provides an example of 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 47-25 describes the syntax.

Table 47-25 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.

How to Use ant to List the Deployed SOA Composite Applications

The following provides an example of 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 47-26 describes the syntax.

Table 47-26 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.

How to Use ant to List All Available Partitions in the SOA Infrastructure

The following provides the syntax for 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 47-27 describes the syntax.

Table 47-27 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.

The following provides an example of listing all available partitions in the SOA Infrastructure:

```
ant -f ant-sca-mgmt.xml listPartitions -Dhost=myhost10 -Dport=8001
```

How to Use ant to List All Composites in a Partition

The following provides the syntax for 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 47-28 describes the syntax.

Table 47-28 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.

The following provides an example of listing all composites in a partition named `myPartition`.

```
ant -f ant-sca-mgmt.xml listCompositesInPartition -Dhost=myhost10 -Dport=8001 -
Dpartition=myPartition
```

How to Use ant to Create a Partition in the SOA Infrastructure

The following provides the syntax for 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 47-29 describes the syntax.

Table 47-29 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`).

Table 47-29 (Cont.) ant Partition Creation 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.
`partition`	The name of the partition to create.

The following provides an example of creating a partition in the SOA Infrastructure named `myPartition`:

```
ant -f ant-sca-mgmt.xml createPartition -Dhost=myhost10 -Dport=8001
-Dpartition=myPartition
```

How to Use ant to Delete a Partition in the SOA Infrastructure

The following provides the syntax for deleting a partition in the SOA Infrastructure. This command undeploys all composites in the partition before deleting the partition.

```
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 47-30 describes the syntax.

Table 47-30 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.

The following provides an example of deleting a partition in the SOA Infrastructure named `myPartition`:

```
ant -f ant-sca-mgmt.xml deletePartition -Dhost=myhost10 -Dport=8001
-Dpartition=myPartition
```

How to Use ant to Start All Composites in the Partition

The following provides the syntax for 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 47-31 describes the syntax.

Table 47-31 ant Partition 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.
`partition`	The name of the partition.

The following provides an example of starting all composites in the partition named `myPartition`:

```
ant -f ant-sca-mgmt.xml startCompositesInPartition -Dhost=myhost10 -Dport=8001
-Dpartition=myPartition
```

How to Use ant to Stop All Composites in the Partition

The following provides the syntax for 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 47-32 describes the syntax.

Table 47-32 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.

The following provides an example of stopping all composites in the partition named `myPartition`:

```
ant -f ant-sca-mgmt.xml stopCompositesInPartition -Dhost=myhost10 -Dport=8001
-Dpartition=myPartition
```

How to Use ant to Activate All Composites in the Partition

The following provides the syntax for 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 47-33 describes the syntax.

Table 47-33 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.

The following provides an example of activating all composites in the partition named `myPartition`:

```
ant -f ant-sca-mgmt.xml activateCompositesInPartition -Dhost=myhost10 -Dport=8001
-Dpartition=myPartition
```

How to Use ant to Retire All Composites in the Partition

The following provides the syntax for 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 47-34 describes the syntax.

Table 47-34 ant Partition Composite Retirement Commands

Argument	Definition
`host`	Hostname of the Oracle WebLogic Server (for example, `myhost`).

Table 47-34 (Cont.) ant Partition Composite Retirement 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.

The following provides an example of retiring all composites in the partition named `myPartition`:

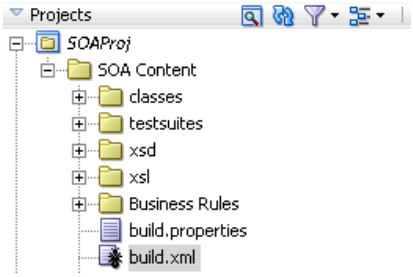
```
ant -f ant-sca-mgmt.xml retireCompositesInPartition -Dhost=myhost10 -Dport=8001
-Dpartition=myPartition
```

How to Use ant to Manage SOA Composite Applications

You can use `ant` scripts to compile, package, and deploy the application. You can create the initial `ant` build files by selecting **New** > **Application** > **Ant** > **Buildfile from Project** from the **File** main menu.

Figure 47-29 shows the **build.properties** and **build.xml** files that display in the Applications window after creation.

Figure 47-29 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.

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 *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

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 *High Availability Guide*.

Deploying SOA Composite Applications with No Servers Running

You can deploy SOA composite applications and shared data (for example, WSDL and XSD files) with no managed SOA servers or administration servers running (known as offline deployment mode). When the servers are restarted, the SOA composite applications and shared data are deployed.

Offline deployment is beneficial for the following use cases:

- Shared data and new SOA composite applications (for example, the system is new and does not have any deployed composites).
- One-off patches that may contain a single SOA composite application (new or patched) or a resource bundle of shared data.

Note the following guidelines when using offline deployment:

- The SOA composite applications and shared data are available in read-only format in the Oracle home directory. You cannot delete or update the composites.
- The same SOA composite application or shared data file can be included in one or all of the supported use cases when offline deployment occurs. However, for a particular SOA composite application, only one composite SAR or shared data file is in the data location relative to the product data root directory. All cases must point to the same root product data directory. The same composite data is overwritten by the order of applied use cases.
- You cannot redeploy or undeploy the SOA composite application through offline deployment.
- Shared data (resource bundle) redeployment is supported since there is no revision concept with shared data.
- WLST commands are provided for adding and removing individual SOA composite applications and shared data to and from offline deployments. For information, see Section "SOA Composite Application Offline Management Deployment" of *WLST Command Reference for SOA Suite*.
- Configuration plans are not supported with offline deployments.

For information about SAR file naming conventions, see Deployed Service Archives.

For information about shared data, see How to Deploy and Use Shared Data Across Multiple SOA Composite Applications in .

Note:

- You cannot deploy ZIP files in offline mode. This is because ZIP files contain other archives.
- You can only deploy a particular composite SAR file into one partition through offline deployment.

Offline Deployment Configuration Files

Two configuration files control offline deployment:

- `soa-configuration.xml` (offline deployment configuration list file).
- `composite-offline-deployments-`*`version_number`*`.xml` (offline deployment configuration file). The *`version_number`* can be any value, but the `composite-offline-deployments-` part is fixed and required.

Offline Deployment Configuration List File

The offline deployment configuration list file identifies the location from which to read the offline deployment configuration files. The file is named `soa-configuration.xml` and appears in the `$DOMAIN/config/fmwconfig` directory. The offline deployment process uses this configuration file to generate a consolidated configuration list to use in offline deployment. The following example shows a `soa-configuration.xml` file in which two directory locations are listed:

```
<?xml version="1.0" encoding="UTF-8"?>
<soa-configuration xmlns="http://xmlns.oracle.com/config/soa">
    <soa-directories>
<soa-directory>/scratch/aime/appTop/common/soa-composiste/soa1</soa-directory>
<soa-directory>/scratch/aime/appTop/common/soa-composiste/soa2</soa-directory>
    </soa-directories>
</soa-configuration>
```

Offline Deployment Configuration File

The offline deployment configuration file specifies the following elements for offline deployments.

- Partitions are created, as necessary, before the SOA composite applications and shared resources are deployed. Note the following order of precedence for partition use:
  - The partition specified in the `<partition>` element is created.
  - If a partition used in `<composite-deployment>` is not specified in the `<partition>` element, it is created implicitly.
  - If the `partition` attribute is not specified in the `<composite-deployment>` element, the composite is deployed into the `default` partition.

- Shared resources

  Shared resources are deployed before the SOA composite applications.

- SOA composite applications

  You can list multiple SOA composite applications in the file. However, they are not deployed in the order in which they are listed in the file.

The file naming convention is `composite-offline-deployments-`*`version_number`*`.xml`, where *`version_number`* can be any value, but the `composite-offline-deployments-` part is fixed and required.

The following example shows the structure of the offline deployment file. The file is divided into the three sections to represent partitions, SOA composite applications, and shared data.

```
<offline-configuration>
    <partitions>?
        <partition name="partition_name"/>*
    </partitions>
    <composite-deployments>?
        <composite-deployment location="/some/path" partition="partition_name"?>*
    </composite-deployments>
    <shared-resources>?
        <shared-resource location="/some/path"/>*
    </shared-resources>
</offline-configuration>
```

The following example shows an offline deployment configuration file in which the following is defined:

- Partition `one` and `two` are created.
- The composite SAR file `/some/path/sca_composite1.jar` is deployed into partition `one`.
- The composite SAR file `/another/path/sca_composite2.jar` is deployed into partition `two`.
- The composite SAR file `/yet/another/path/sca_composite3.jar` is deployed into the `default` partition.
- The shared data JAR files `/some/path/shareddata1.jar` and `/another/path/shareddata2.jar` are deployed into the shared data location.

```
<offline-configuration>
    <composite-deployments>
        <composite-deployment location="/some/path/sca_composite1.jar"
 partition="one">
        <composite-deployment location="/another/path/sca_composite2.jar"/
 partition="two">
        <composite-deployment location="/yet/another/path/sca_composite3.jar"/>
    </composite-deployments>
    <shared-resources>
        <shared-resource location="/some/path/shareddata1.jar"/>
        <shared-resource location="/another/path/shareddata2.jar"/>
    </shared-resources>
</offline-configuration>
```

The following example shows an offline configuration deployment file in which the following occurs:

- The shared data JAR file named `shareddata.jar` is deployed.
- The composite SAR file named `sca_soaApp1.jar` is deployed into the `myPartition` partition.
- The composite SAR file named `sca_soaApp2.jar` is deployed by default into the `default` partition because no partition is explicitly defined.

```
<offline-configuration>
    <composite-deployments>
        <composite-deployment
            location="/scratch/aime/appTop/soa1/sca_soaApp1.jar“
            partition=“myPartition“/>
        <composite-deployment
            location="/scratch/aime/appTop/soa1/sca_soaApp2.jar“/>
           </composite-deployments>
    <shared-resources>
        <shared-resource location="/scratch/aime/appTop/soa1/shareddata.jar"/>
    </shared-resources>
</offline-configuration>
```

The following example shows an offline deployment configuration file in which only shared data located in the two defined directories is deployed:

```
<offline-configuration>
    <shared-resources>
        <shared-resource location="/some/path/shareddata1.jar"/>
        <shared-resource location= "/another/path/shareddata2.jar"/>
    </shared-resources>
</offline-configuration>
```

The following example shows an offline deployment file in which partition `one` and `two` are created. No SOA composite applications or shared data are deployed:

```
<offline-configuration>
    <partitions>
        <partition name="one"/>
        <partition name="two"/>
    </partitions>
</offline-configuration>
```

Relative Configuration File Paths

Relative paths are also supported in the offline deployment configuration file. The following example shows the `soa-configuration.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<soa-configuration xmlns="http://xmlns.oracle.com/config/soa">
    <soa-directories>
<soa-directory>/scratch/aime/appTop/common/soa-composiste/soa1</soa-directory>
    </soa-directories>
</soa-configuration>
```

The following example shows the offline deployment `composite-offline-deployments-1.0.xml` file. The two composite SAR files and one shared data JAR file are all located in the `soa1` directory shown in the preceding example.

```
<offline-configuration>
    <composite-deployments>
        <composite-deployment location="sca_soaApproval.jar" >
        <composite-deployment location="sca_soaNotification.jar">
    </composite-deployments>
```

```
    <shared-resources>
        <shared-resource location="soashareddata.jar"/>
    </shared-resources>
</offline-configuration>
```

Order of Deployment

Offline deployments are processed in the following order:

- The `soa-configuration.xml` offline deployment configuration list file is read to identify the location of the offline deployment configuration files (`composite-offline-deployments-`*`version`*`.xml`).
- The `composite-offline-deployments-`*`version`*`.xml` files are read and a consolidated list is created based on the file location. The consolidated list contains the partitions, shared data files, and SOA composite application files.
- The consolidated list is processed in the following order:
  - Partitions
  - Shared data files
  - SOA composite application files

How to Deploy SOA Composite Applications and Shared Data with No Server Running

This section provides an overview of the procedures for deploying SOA composite applications and shared data with no server running.

To deploy SOA composite applications and shared data with no server running:

1. Create an offline deployment configuration list file. This file identifies the location from which to read the offline deployment configuration files. For examples of the contents of this file, see Offline Deployment Configuration List File.
2. Create offline deployment configuration files. This file specifies the elements to include in the offline deployment (partitions, shared data, or SOA composite applications). For examples of the contents of this file, see Offline Deployment Configuration File.
3. Restart the SOA servers.

   The composites are deployed and displayed in the **Deployed Composites** tab of the SOA Infrastructure in Oracle Enterprise Manager Fusion Middleware Control. For more information, see "Managing the State of All Applications at the SOA Infrastructure Level" of *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

   If troubleshooting is required, you can view deployment results in the SOA server diagnostic log file.

What You May Need to Know About Offline Composite Deployment in a Cluster Environment

When the server starts up during offline composite deployment, the SOA composite application is deployed to all nodes in the cluster. The registration files are supported in one physical domain location, rather than synchronizing the files across all physical domain locations in the cluster. If the cluster is configured where the domain location

is present on different physical hosts, select the domain directory on one host and use that as the offline registration location.

What You May Need to Know About Deploying SOA Composite Applications that Reference Shared Data That is Not in the MDS Repository

Offline deployment enables a SOA composite application that references shared artifacts in the MDS Repository to be deployed when the shared data is not present in the MDS Repository.

This is the expected behavior. To save time during server startup, offline deployment uses lazy loading by default. With lazy loading, you do not see a deployment error when the composite is deployed during server startup if the composite is referencing nonexistent shared data. However, you do see the failure when you invoke the composite for the first time. The composite fails if it references non-existent, shared data. With lazy loading, the failure point is different; it is not in the deployment, but in the first invocation.

Postdeployment Configuration

This section describes postdeployment configuration tasks.

Security

For information about securing SOA composite applications, see *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

Updating Connections

Ensure that any connections that you created to the application server or MDS Repository are recreated to point to servers applicable to the next target environment. For more information, see Creating an Application Server Connection and Creating a SOA-MDS Connection.

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 How to Create Data Sources and Queues and How to Create Connection Factories and Connection Pooling.

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 *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

Testing and Troubleshooting

This section describes how to test and troubleshoot your SOA composite application.

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 *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

Initiating an Instance of a Deployed Composite

You can initiate an instance of a deployed SOA composite application from the Test Web Service page in Oracle Enterprise Manager Fusion Middleware Control. For more information, see *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

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 run from either Oracle JDeveloper or Oracle Enterprise Manager Fusion Middleware Control.

For information about creating and running test cases from Oracle JDeveloper, see Automating Testing of SOA Composite Applications.

For information about running test cases from Oracle Enterprise Manager Fusion Middleware Control, see *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

Recompiling a Project After Receiving a Deployment Error

If you receive the error shown in the following example 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.

```
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)
```

Reducing Java Code Size to Resolve Java Compilation Errors

If you receive the Java compilation error shown in the following example in your server log files, you may have too much code in your Java classes.

```
Failed to compile bpel generated classes.
failure to compile the generated BPEL classes for BPEL process
"Review_Supply_Plan_ProcessProcess" of composite "default/Review_Supp
```

```
ly_Plan_Process!1.0*a9ca2907-8540-4375-b672-ceb560d7b826"
The class path setting is incorrect.
Ensure that the class path is set correctly. If this happens on the server
side, verify that the custom classes or jars which this BPEL process is
depending on are deployed correctly. Also verify that the runtime is using
the same release/version.
. . .
. . .
        at
com.collaxa.cube.lang.compiler.template.CubeProcessGenerator.compile(CubeProce
ssGenerator.java:304)
        at
com.collaxa.cube.lang.compiler.template.CubeProcessGenerator.generate(CubeProc
essGenerator.java:164)
        at
com.collaxa.cube.lang.compiler.BPEL1Processor.transform(BPEL1Processor.java:25
7)
        at
com.collaxa.cube.lang.compiler.BPEL1Processor.process(BPEL1Processor.java:161)
```

To reduce Java code size to resolve Java compilation errors:

1. Open the `$MIDDLEWARE_HOME/user_projects/domains/`*`domain_name`*`/bin/SetDomainEnv.sh` file (for Linux) or `SetDomainEnv.bat` file (for Windows).
2. Locate the `EXTRA_JAVA_PROPERTIES="-Dorabpel.codegen.density"` property in this file. If this property is not explicitly set, it defaults to values of `64,32`.
3. Reduce the values:

   ```
   EXTRA_JAVA_PROPERTIES="-Dorabpel.codegen.density=32,16"
   ```

   By reducing these two values, you increase the number of classes and methods that are generated for the compiled process map. As a best practice, if the process fails to compile using the default settings, set the property with smaller values. The following values are good combinations to try:

   ```
   32,16
   16,8
   8,4
   4,2
   ```

4. Save your changes.
5. Restart the server.
6. Recompile your SOA composite application.

Troubleshooting Common Deployment Errors

This section describes how to troubleshoot common deployment errors.

For information about general composite application troubleshooting issues, see *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

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 Applications window, 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 Applications window, 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** check box 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 the following example.

  ```
  Compilation of project 'FirstComposite.jpr' finished. Check '/scratch/myhome/
  jdevWorkarea/mywork/Application11/FirstComposite/SCA-INF/classes/scac.log' for
  details.
  ```

- After compilation is successful, a SAR/SOA bundle archive is built for the composite. For a SAR archive, the message shown in the following example is displayed in the **Deployment** tab.

  ```
  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 the following example is displayed in the **Deployment** tab.

  ```
  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 the following example in the **Deployment** tab.

  ```
  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 the following example is displayed in the **Deployment** tab with an error message (if any) from the server.

  ```
  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`.

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. The following example provides details.

  ```
  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.

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 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.

You can use WLST to deploy SOA composite applications to a managed Oracle WebLogic Server without starting an Oracle WebLogic Administration Server. See Deploying and Managing SOA Composite Applications with the WLST Utility for details.

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.

Deploying with an Unreachable Proxy Server

You can receive an error similar to that shown in Figure 47-30 during SOA composite application deployment if you have a proxy server set in Oracle JDeveloper that is not reachable from your host.

Figure 47-30 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.

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 error means you must first release the lock from Oracle WebLogic Server Administration Console to successfully deploy the EAR file.

To release locks to resolve ADF task form EAR file deployment errors:

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.
   ```

This error can occur regardless of the deployment method you are using (for example, deploying through Oracle JDeveloper or through `ant` scripts).

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. Open the `ant-sca-compile.xml` file in the `$ORACLE_HOME/bin` directory.
2. Under the `scac` element, increase the memory setting. For example:

   ```
   <jvmarg value="-Xmx512M"/>
   ```

Oracle JDeveloper Compilation Error When Property Alias Definition is Missing for a Receive Activity with a Correlation Set

When a property alias definition is missing for a receive activity with a correlation set, the Oracle JDeveloper compiler fails with `SCAC-50012` error.

ADF Binding Service Names Must Be Unique Across All Deployed SOA Composite Applications

All ADF bindings must have a unique service name across all deployed SOA composite applications.

For example, assume you perform the following steps:

1. Build and successfully deploy a SOA composite application that includes multiple composites to the SOA server.
2. Change one of the composites in the SOA composite application by adding new components and an outbound external reference.
3. Compile and successfully build the updated SOA composite application as revision 2.0.
4. Deploy the updated SOA composite application to the same partition or a different partition.

   You can receive the following error:

   ```
   <Oct 7, 2013 11:52:01 AM EDT> <Error> <ServletContext-/soa-infra>
   <BEA-000000> <Error during deployment
   oracle.fabric.common.FabricException: Deployment Failed: The serviceName
   ```

```
attribute "OrderProcessorService" has already been used.   ServiceName must
be unique among all deployed composites.  The new service will overwrite the
old one.
        at
oracle.integration.platform.blocks.deploy.StandaloneCompositeDeploymentCoordin
atorImpl.coordinateCompositeDeployment(StandaloneCompositeDeploymentCoordinato
rImpl.java:99)
. . .
. . .
```

This error occurred because all ADF bindings must have a unique service name across all deployed SOA composite applications.

5. As a workaround, you must edit the `composite.xml` file and assign a different name for the service in the.adf `binding.adf` section. For example:

```
<binding.adf serviceName="OrderProcessorService_v2" registryName=""/>
    <!-- exposed for using via direct binding api -->
```

48

Using the Oracle SOA Suite Development Maven Plug-In

This chapter describes how to use the Oracle SOA Suite development Maven plug-in to build and manage SOA composite application projects. The Oracle SOA Suite development Maven plug-in enables you to compile, package, deploy, test, and undeploy a SOA composite application in a Maven environment.

This chapter includes the following sections:

For more information about using Maven with Oracle Fusion Middleware, see *Developing Applications Using Continuous Integration* and Section "Using the WebLogic Development Maven Plug-In" of *Developing Applications for Oracle WebLogic Server*.

For detailed information on using Maven to build applications and projects, see `http://maven.apache.org/users/index.html`.

Introduction to the Oracle SOA Suite Maven Plug-in

Maven is a build automation tool that enables you to create and manage runtime projects. Using the Oracle SOA Suite Maven plug-in, you can build and manage a SOA composite application. Maven relies on an artifact repository for all of its dependencies. All the installed Oracle libraries are propagated into the Maven repository. This enables Maven to recognize them as artifacts and address them in the Project Object Model (POM) file.

POM Files and Archetypes

Maven projects are configured using a POM file. The POM file describes dependencies such as the SOA Infrastructure tools that are required to build the composites.

An archetype is a template for creating a project. Archetypes are provided to create a new SOA application containing a single SOA project, or to add an additional SOA project to an existing SOA application. These archetypes provide for the ability to compile, package, deploy, test, and undeploy a SOA composite application.

The following shows a sample Maven POM file for Oracle SOA Suite:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
 http://maven.apache.org/xsd/maven-4.0.0.xsd"
 xmlns="http://maven.apache.org/POM/4.0.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <modelVersion>4.0.0</modelVersion>
```

```
<groupId>com.test</groupId>
<artifactId>MyComposite</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>sar</packaging>

<!--
       This POM was generated from the SOA Maven Archetype.
       Comments in this POM guide you how to use it with your project.
       This POM relates to this SOA Composite, i.e. the one in this same
       directory. There is another POM in the SOA Application directory (up
       one) which handles the whold SOA Application, which may contain
       additional projects.

       The parent points to the common SOA parent POM.  That is a special POM
       that is shipped by Oracle as a point of customization (only).  You can
       add default values for properties like serverUrl, etc. to the SOA
       common parent POM, so that you do not have to specify them over and
       over in every project POM.
-->
<parent>
    <groupId>com.oracle.soa</groupId>
    <artifactId>soa-project-common</artifactId>
    <version>12.1.3-0-0</version>
</parent>

<properties>
    <!-- these parameters are used by the compile goal -->
    <scac.input.dir>${project.basedir}/SOA/</scac.input.dir>
    <scac.output.dir>${project.basedir}/target</scac.output.dir>
    <scac.input>${scac.input.dir}/composite.xml</scac.input>
    <scac.output>${scac.output.dir}/out.xml</scac.output>
    <scac.error>${scac.output.dir}/error.txt</scac.error>
    <scac.displayLevel>1</scac.displayLevel>
    <!-- if you are using a config plan, uncomment the following line and
         update to point to your config plan -->
    <!--<configplan>${scac.input.dir}/configplan.xml</configplan>-->

    <!-- these parameters are used by the deploy and undeploy goals -->
    <composite.name>${project.artifactId}</composite.name>
    <composite.revision>${project.version}</composite.revision>
    <composite.partition>default</composite.partition>
    <serverUrl>serverUrl</serverUrl>
    <user>user</user>
    <password>password</password>
    <overwrite>true</overwrite>
    <forceDefault>true</forceDefault>
    <regenerateRulebase>false</regenerateRulebase>
    <keepInstancesOnRedeploy>false</keepInstancesOnRedeploy>

    <!-- these parameters are used by the test goal -->
    <!-- if you are using the sca-test (test) goal, you need to uncomment the
         following line and point it to your jndi.properties file. -->
    <jndi.properties.input>${basedir}/jndi.properties</jndi.properties.input>
    <scatest.result>${scac.output.dir}/testResult</scatest.result>
    <!--  input is the name of the composite to run test suties against -->
        <input>MyComposite</input>
</properties>

<!--
       These refer to the properties defined above.  You should probably not
       need to make any changes beflow this point - these just point to the
```

```
            properties above.
        -->
        <build>
            <plugins>
                <plugin>
                    <groupId>com.oracle.soa.plugin</groupId>
                    <artifactId>oracle-soa-plugin</artifactId>
                    <version>12.1.3-0-0</version>
                    <configuration>
                        <compositeName>MyComposite</compositeName>
                        <composite>${scac.input}</composite>
                        <sarLocation>${scac.output.dir}/sca_${project.artifactId}_
                                    rev${composite.revision}.jar</sarLocation>
                        <serverUrl>${serverUrl}</serverUrl>
                        <!-- note: compositeRevision is needed to package, revision is
                             needed to undeploy -->
                        <compositeRevision>${composite.revision}</compositeRevision>
                        <revision>${composite.revision}</revision>
                        <scacInputDir>${scac.input.dir}</scacInputDir>
                        <!-- note: if this composite contains a component that depends
                             on MDS to build, e.g. a Human Task or Business Rule, then
                             you will need to uncomment the next line, and edit it to
                             point to your application directory (which contains
                             .adf/adf-config.xml file with MDS configuration in it -->
                        <!--<appHome>${project.basedir}/..</appHome>-->
                        <!--  If you have a composite which contains a component that
                              depends on MDS (eg. Human Task, Business Rule) AND you
                              want to use a file-based MDS repository, then you
                              need to do either:
                              1. update the .adf/META-INF/adf-config.xml to point to
                                 the correct location of the file based repository,
                                 i.e. remove the reference to ${oracle.home} in that
                                 file, or
                              2. define oracleHome here and leave the adf-config.xml
                                 file as is.  Note that the correct value is the path
                                 to your SOA Quickstart or JDeveloper install
                                 directory, with "/soa" appended to it.
                        -->
                        <!--<oracleHome>JDEV_HOME/soa</oracleHome>-->
                        <user>${user}</user>
                        <password>${password}</password>
                                <input>${input}</input>
                    </configuration>
                    <!-- extensions=true is needed to use the custom sar packaging
                         type -->
                    <extensions>true</extensions>
                </plugin>
            </plugins>
        </build>
    </project>
```

Note:

If you are using a component in your composite that depends on MDS, such as Human Tasks or Business rules, you must uncomment the `<appHome>${project.basedir}/..</appHome>` line and edit it to point to your application directory.

If you are using a component that depends on MDS and you want to use a file-based MDS, such as the one referenced in the default `adf-config.xml` file, you must also uncomment the `<oracleHome>JDEV_HOME/soa</oracleHome>` line and edit it to point to your SOA Quickstart or JDeveloper install directory, with `/soa` appended to it.

The following shows the archetype coordinates in the POM file for creating an Oracle SOA Suite Application:

```
<groupId>com.oracle.soa.archetype</groupId>
<artifactId>oracle-soa-application</artifactId>
<version>12.1.3-0-0</version>
```

Maven Plug-in Goals

Goals are associated with different phases of the composite life cycle.

When you invoke a goal associated with a life cycle phase, Maven executes all goals mapped to all phases up to and including the goal you name. For example, if you execute the test goal, the compile, package, and deploy goals are executed before the test goal. The description of each goal in this section lists the actions performed when each goal is invoked.

To support the life cycle of building and deploying a SOA composite application, the following executable plug-in goals are provided.

- compile (scac)
- package (sar)
- deploy
- test (sca-test)
- undeploy

Before executing a goal, ensure that you have provided all of the necessary parameters for that goal in the POM file. See POM Files and Archetypes for a sample POM file.

Note:

If you have changed the name of the project, composite, or project directory, ensure that you update the POM file with the new names before executing any of these goals.

The following example shows the `groupId`, `artifactId`, and `version` coordinates for Oracle SOA Suite plug-ins in the POM file.

```
<groupId>com.oracle.soa.plugin</groupId>
<artifactId>oracle-soa-plugin</artifactId>
<version>12.1.3-0-0</version>
```

compile

The compile goal compiles a SOA composite application. Oracle SOA Suite provides a native Maven implementation for this goal. The following command compiles the SOA composite application:

```
mvn compile
```

package

The package goal packages the artifacts of a SOA composite application into a SOA archive (SAR) file. The following command compiles and packages the SOA composite application:

```
mvn package
```

deploy

The deploy goal deploys the SOA composite application. Oracle SOA Suite provides a native Maven implementation for this goal. The following command compiles the SOA composite application, packages the composite into a SAR file, and deploys the SAR file to the server.

```
mvn pre-integration-test
```

test

The test goal performs a test of a SOA composite application. Oracle SOA Suite provides a native Maven implementation for this goal.

You must first create tests in Oracle JDeveloper before running the test goal. For more information about creating tests using JDeveloper, see Automating Testing of SOA Composite Applications.

You must also include a `jndi.properites` file before running the test goal. Edit the following line in the POM file to point to a `jndi.properties` file:
<jndi.properties.input>${basedir}/jndi.properties</jndi.properties.input>

The following shows a sample `jndi.properties` file:

```
java.naming.factory.initial=weblogic.jndi.WLInitialContextFactory
java.naming.provider.url=t3://servername:7103/soa-infra
java.naming.security.principal=weblogic
java.naming.security.credentials=welcome1
dedicated.connection=true
dedicated.rmicontext=true
```

The following command compiles the composite, packages the composite into a SAR file, deploys the SAR file to the server, and tests the composite.

```
mvn verify
```

undeploy

The undeploy goal undeploys the SOA composite application. Oracle SOA Suite provides a native Maven implementation for this goal. The following command undeploys the composite.

Note:

The undeploy goal is not mapped to a life cycle phase. You must explicitly invoke it by name.

```
mvn com.oracle.soa.plugin:oracle-soa-plugin:undeploy
```

Using Maven Online Help

Maven online help provides you with a list of goals and their associated commands. For example, enter the following command to obtain online help for the Maven test goal:

```
mvn help:describe -Ddetail=true -Dplugin=com.oracle.soa.plugin:oracle-soa-plugin:
12.1.3-0-0 -Dgoal=test
```

This command displays the following help details:

```
oracle-soa:test
Description: Description: To execute SCA Test Suites.
Implementation: com.oracle.soa.plugin.SoaTest
Language: java
Bound to phase: verify
Goal Prefix: oracle-soa

Available parameters:

   format (Default: native)
      User property: format
      The format of the output - 'native' or 'junit'.

   input
      Required: true
      User property: input
      The name of the composite to execute tests against.

   jndiPropertiesInput
         Required: true
         User property: jndi.properties.input
         Path to JNDI properties file required for SCA Test execution.
         This file should contain contents similar to the following:
         java.naming.factory.initial=weblogic.jndi.WLInitialContextFactory
         java.naming.provider.url=t3://servername:7103/soa-infra
         java.naming.security.principal=weblogic
         java.naming.security.credentials=welcome1
         dedicated.connection=true
         dedicated.rmicontext=true

      partition (Default: default)
         User property: partition
         Which SOA partition the composite is deployed in.

      result (Default: ${java.io.tmpdir}/out)
         User property: result
         Where to place the results.

      timeout (Default: 300)
         User property: timeout
         How long to wait for tests to complete before timing out.
```

Installing the Oracle SOA Suite Maven Plug-in

A distribution of Maven 3.0.5 is included with Oracle Fusion Middleware in the following location:

Middleware_Home/*Oracle_Home*/oracle_common/modules/org.apache.maven_3.0.5

For information about installing Maven for Oracle Fusion Middleware, see "Installing and Configuring Maven for Build Automation and Dependency Management" in *Developing Applications Using Continuous Integration*. Be sure to follow the setup instructions in Section 5.1, "Setting Up the Maven Distribution" and Section 5.2, "Customizing Maven Settings."

How to Configure the Oracle SOA Suite Maven Plug-In

Before you can use the Oracle SOA Suite Maven plug-in you must populate the Maven repository with Oracle artifacts. For more information about populating the repository, see "Populating the Maven Repository Manager" in *Developing Applications Using Continuous Integration* for more information. The steps below link to specific sections of this guide.

To configure the Oracle SOA Suite development Maven plug-in:

1. Navigate to *ORACLE_HOME*/oracle_common/plugins/maven/com/oracle/maven/oracle-maven-sync/12.1.3.

2. Run the following command to install the Maven sync plug-in:

   ```
   mvn install:install-file -DpomFile=oracle-maven-sync-12.1.3.pom -Dfile=oracle-maven-sync-12.1.3.jar
   ```

   For more options, see "Installing Oracle Maven Synchronization Plug-In."

3. Run the following command to seed the Oracle SOA Suite development Maven plug-in into the Maven repository:

   ```
   mvn com.oracle.maven:oracle-maven-sync:push -DoracleHome=ORACLE_HOME
   ```

   Where *ORACLE_HOME* is the full path to your Oracle Fusion Middleware installation. For more options, see "Running the Oracle Maven Synchronization Plug-In."

4. Validate whether you have successfully installed the plug-in using the Maven `help:describe` goal.

   ```
   mvn help:describe -DgroupId=com.oracle.soa.plugin
   -DartifactId=oracle-soa-plugin -Dversion=12.1.3-0-0
   ```

   The following is an excerpt of the information that confirms installation of the Oracle SOA Suite plug-in:

   ```
   Name: Oracle SOA Maven Plugin
   Description: This plugin allows users to compile, package, deploy, test and
   undeploy SOA composites.
   Group Id: com.oracle.soa.plugin
   Artifact Id: oracle-soa-plugin
   Version: 12.1.3-0-0
   Goal Prefix: oracle-soa
   ```

```
This plugin has 6 goals:
oracle-soa:compile
```

Using the Oracle SOA Suite Maven Archetype

Use the Oracle SOA Suite archetype to generate a POM file for a SOA application. Run the following command from the parent directory into which you want to add a SOA application. The SOA application is created in a subdirectory named from the value of the `artifactId` property.

> **Note:**
>
> SOA Applications created using the Oracle SOA Suite Maven archetype are the same as those created in Oracle JDeveloper using the Create SOA Application wizard.

```
mvn archetype:generate
 -DarchetypeGroupId=com.oracle.soa.archetype
 -DarchetypeArtifactId=oracle-soa-application
 -DarchetypeVersion=12.1.3-0-0
 -DarchetypeRepository=local
 -DgroupId=org.my.test
 -DartifactId=test-soa-application
 -DprojectName=test-soa-project
 -Dversion=1.0-SNAPSHOT
```

Where:

Property	Description
`archetypeGroupId`	Enter the group ID of the archetype to use (`com.oracle.soa.archetype`).
`archetypeArtifactId`	Enter the artifact ID of the archetype to use (`oracle-soa-application`).
`archetypeVersion`	Enter the archetype version (`12.1.3-0-0`).
`archetypeRepository`	Enter the Maven repository to use. (Optional)
`groupId`	Enter the group ID of the project to build (for this example, `org.my.test`).
`artifactId`	Enter the artifact ID of the project to build. This becomes the name of the subdirectory (for this example, `test-soa-project`) in the current directory. The SOA application and the first SOA project are created in this subdirectory.
`projectName`	Enter the name for the SOA Project to be created inside the SOA application This is also the name of the composite.
`package`	Enter the name for the SOA Project to be created inside the SOA application. (Optional)

Property	Description
`version`	Enter the version of the project to build (for this example, `1.0-SNAPSHOT`)

49

Debugging and Auditing SOA Composite Applications

This chapter describes how to debug SOA composite applications with the SOA debugger in Oracle JDeveloper, test HTTP requests and response messages in the HTTP Analyzer, and configure auditing for BPEL process activities in a SOA composite application.

This chapter includes the following sections:

Introduction to the SOA Debugger

You can test and debug SOA composite applications with the SOA debugger in Oracle JDeveloper. The SOA debugger reduces the development cycle for a SOA composite application by providing a troubleshooting environment within Oracle JDeveloper. This eliminates the lengthy process of building a SOA composite application in Oracle JDeveloper, deploying it to the SOA Infrastructure, starting Oracle Enterprise Manager Fusion Middleware Control to test or view audit trails and flow traces, and then returning to Oracle JDeveloper to repeat the exercise. Instead, you can set breakpoints in Oracle JDeveloper for troubleshooting on the following components:

- Binding components and service components in SOA composite applications
- Synchronous and asynchronous BPEL processes
- Oracle BPM processes
- Oracle Service Bus pipelines (See Section "Debugging Oracle Service Bus Applications" of *Developing Services with Oracle Service Bus*)

Note the following guidelines when using the SOA debugger:

- The SOA composite application name and the Oracle JDeveloper project name *must* be the same.
- Any SOA composite application encountered during a debugging session must reside in the currently active workspace in Oracle JDeveloper.
- Debugging is limited to design view in Oracle JDeveloper. You cannot run the SOA debugger in Oracle Enterprise Manager Fusion Middleware Control.

- Debugging is a localized user experience. If you want to switch to other tasks (for example, search for instances or initiate new instances of the same composite from Oracle Enterprise Manager Fusion Middleware Control), close the debugging session.
- You cannot set breakpoints on REST services.
- The breakpoints that you create for debugging in a SOA composite application in one installation of Oracle JDeveloper are not available to other Oracle JDeveloper installations. You must create the breakpoints again for debugging.
- During a debugging session in which you are using the embedded Integrated WebLogic Server, you cannot use the version of Oracle Enterprise Manager Fusion Middleware Control included with the embedded server to generate new instances or query instances. For information about the Integrated WebLogic Server, see *Installing SOA Suite and Business Process Management Suite Quick Start for Developers*.
- You cannot debug cross-language features, such as a Java `exec` activity, XSLT and XQuery transformations, and so on.
- You can debug SOA composite applications on servers on which Oracle SOA Suite is installed. For example, if Oracle SOA Suite runs on managed servers, clients must connect using the managed server host and port.
- Only one client at a time can connect to the SOA debugger.
- You cannot debug multiple instances of the same SOA composite application at a given time even though Oracle JDeveloper does not restrict you from this action. This is not the correct approach. The SOA debugger is a development tool. It is your responsibility to ensure that only a single instance is debugged at any given time.
- Adapter endpoint errors are not displayed in the SOA debugger in Oracle JDeveloper. Those errors are logged in the log file.
- You can only debug if the server is in development mode. Debugging in production mode is not supported.
- Oracle B2B and Oracle SOA for Healthcare service and reference binding components *cannot* be debugged with the SOA debugger even though you can set debugging points on both components.
- SOA composite application-to-SOA composite application debugging is not supported.

Debugging a SOA Composite Application

This section describes how to create breakpoints and debug SOA composite applications in Oracle JDeveloper.

Note:

Do not attempt to debug SOA composite applications with very large payloads. Attempting to do so results in the SOA debugger breakpoints hanging in the outbound direction.

How to Start the SOA Debugger

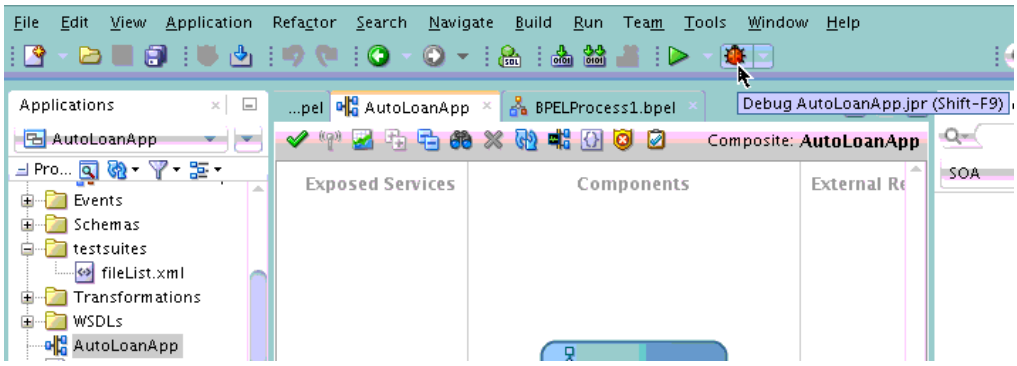
To start the SOA debugger:

1. Start the Integrated WebLogic Server. For information about starting the Integrated WebLogic Server with the **Start Server Instance** option, see Section "Installing Oracle SOA Suite Quick Start for Developers" of *Installing SOA Suite and Business Process Management Suite Quick Start for Developers*.

2. Start the SOA debugger in either of the following ways. This is limited to single composite debugging.

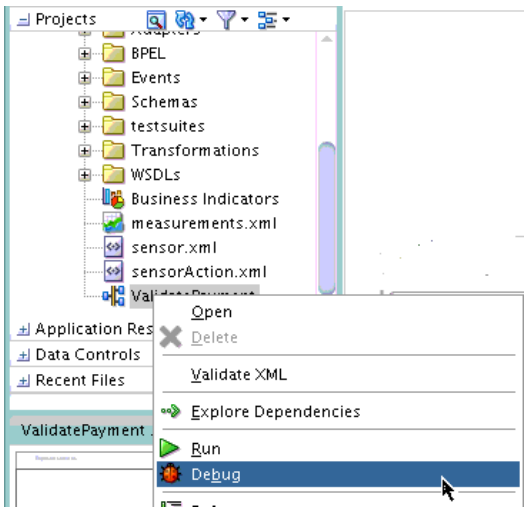
   a. Click the debugger icon above the SOA Composite Editor, as shown in Figure 49-1.

   Figure 49-1 Debugger Icon in SOA Composite Editor

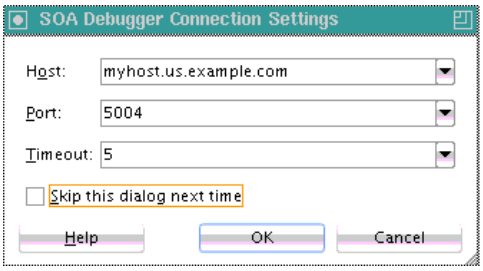
   b. Right-click the SOA composite application in the Applications window, and select **Debug**. Figure 49-2 provides details.

   Figure 49-2 Debug Menu Option for a SOA Composite Application in the Applications Window

   The SOA Debugger Connection Settings dialog is displayed, as shown in Figure 49-3. This dialog enables you to define the SOA debugging server to use.

Figure 49-3 SOA Debugger Connection Settings Dialog



3. Enter values appropriate to your environment, and click **OK**. Table 49-1 provides details.

Table 49-1 SOA Debugger Connection Setting Dialog

Field	Description
Host	Select the debugging server to which to connect. By default, the name of the local host is displayed. This is the embedded Integrated WebLogic Server in Oracle JDeveloper. You can also enter a remote server. When a project is imported from a different host, the host that was used there is displayed here.

Port	Enter the port on which the debugging agent listens. The default value is `5004`. Debugging is automatically enabled in development environments when you install the Oracle SOA Suite Developer Quick Install. The debugger cannot be enabled in production mode or when the server is part of a cluster. For development environments, the debugger can be overridden by adding the following property settings in the `setDomainEnv.sh` file.
	`export SOA_DEBUG_FLAG="true"`
	`export SOA_DEBUG_PORT="5004"`

Timeout	Specify in minutes how long the client should wait while attempting to establish a debugging session before stopping. The default value is `5` minutes. For synchronous processes, you can increase the default value: • Increase the **SyncMaxWaitTime** property in Oracle Enterprise Manager Fusion Middleware Control. For more information, see How To Specify Transaction Timeout Values. • Increase the **Idle Timeout** and **Transaction Timeout** values for the Enterprise JavaBeans property **BPELDeliveryBean** in Oracle WebLogic Server Administration Console. For information about accessing these properties, see the "Long Running, Synchronous Calls To Remote Web Services Error Out or Asynchronous Transactions Return with an Error after a Long Time" section of *Administering Oracle SOA Suite and Oracle Business Process Management Suite*. • Increase the Java Transaction API (JTA) timeout value located under the **JTA** link on the Oracle WebLogic Server Administration Console home page.

Table 49-1 (Cont.) SOA Debugger Connection Setting Dialog

Field	Description
Skip this dialog next time	Select to skip this dialog the next time you begin a debugger session. The settings you previously defined are used. You can display this dialog again by right-clicking the project in the Applications window. Select **Project Properties** > **Run/Debug** > **Edit** > **Tool Settings** > **Debugger** > **Remote**, and select the **Show Dialog Box Before Connecting Debugger** check box.

Note:

You can also edit these properties by right-clicking the project in the Applications window, and selecting **Project Properties** > **Run/Debug** > **Edit** > **Tool Settings** > **Debugger** > **Remote**.

A check is made to determine if the SOA composite application selected for debugging is deployed. Table 49-2 provides details.

Table 49-2 Check to Determine if the SOA Composite Application is Deployed

If the SOA Composite Application Is...	Then...
Deployed	The following message is displayed on the right handle of the service binding component: `Use context menu to initiate WS debugging` See Figure 49-5 for an example of this message. You are ready to begin debugging. Go to How to Set Breakpoints and Initiate Debugging.

Table 49-2 (Cont.) Check to Determine if the SOA Composite Application is Deployed

If the SOA Composite Application Is...	Then...
• Not deployed • Deployed, but there has been a design change in the composite since it was deployed. **Note**: Composites deployed a second time with the **Overwrite any existing composites with the same revision ID** check box selected do not require an additional redeployment. • Deployed, but you removed the Oracle JDeveloper system folder. The system folder is identified by selecting **Help** > **About** > **Properties**, and searching for **ide.system.dir**. • Deployed in one Oracle JDeveloper, but the ZIP file of the SOA composite application was opened in a different installation of Oracle JDeveloper.	The Deployment Action page of the Deploy *Project_Name* wizard is displayed, and you must deploy the composite. **a.** Select **Deploy to Application Server**. **b.** Follow the pages of the wizard to deploy the SOA composite application to an application server. For information about deploying SOA composite applications, see Deploying the Profile. **c.** When deployment is complete, go to How to Set Breakpoints and Initiate Debugging.

You are ready to begin a debugging session when the following message is displayed in the Log window:

```
Debugger attempting to connect to remote process at host_name 5004
Debugger connected to remote process at host_name 5004
Debugger process virtual machine is SOA Debugger.
```

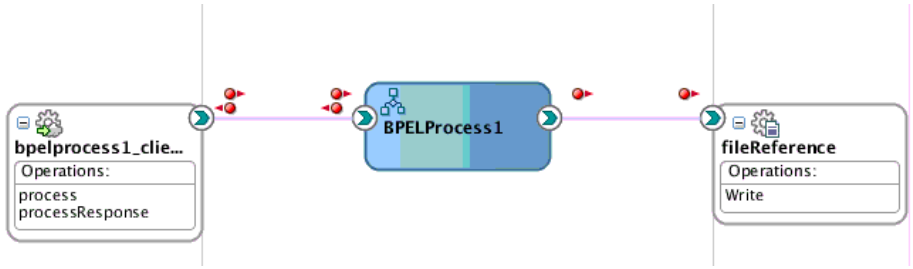
How to Set Breakpoints and Initiate Debugging

Breakpoints are the intentional pausing locations in a SOA composite application that you set for debugging purposes. You can set breakpoints on the following components:

- Service binding components
- Inbound and outbound parts of BPEL process activities and BPM process service components
- Reference binding components such as web services and JCA adapters
- Oracle Service Bus services (see "Debugging Oracle Service Bus Applications" of *Developing Services with Oracle Service Bus*)

Components on which breakpoints are set are designated with red request (outbound) icons, reply (inbound) icons, or request-reply (outbound-inbound) icons. Figure 49-4 provides an example of a SOA composite application in which breakpoint icons have been set.

Figure 49-4 SOA Composite Application with Breakpoints Set



To set breakpoints and initiate debugging:

1. Select the component on which to set the breakpoint, as shown in Table 49-3.

Table 49-3 Components on Which to Set Breakpoints

To Set a Breakpoint on a...	Go to Step...
Service binding component	2
Reference binding component	3
Service component such as a BPEL process or BPM process	4

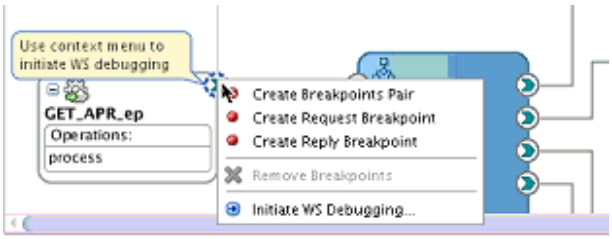
2. To set a breakpoint on a service binding component.

   a. Right-click the right handle of the service on which the following message is displayed.

   ```
   Use context menu to initiate WS debugging
   ```

   This action invokes the context menu shown in Figure 49-5.

Figure 49-5 SOA Debugger Breakpoint Menu Options



   b. Select the appropriate breakpoint interaction option shown in Table 49-4.

Table 49-4 Breakpoint Interaction Options

Option	Description
Create Breakpoints Pair	Set for a request-reply (outbound-inbound) interaction. This is useful for scenarios in which both the request and reply are important.
Create Request Breakpoint	Set for a request (outbound) interaction. This is useful for scenarios in which only the request is important.

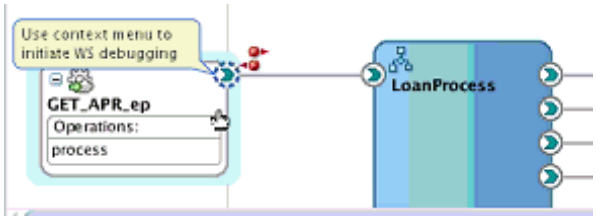
Table 49-4 (Cont.) Breakpoint Interaction Options

Option	Description
Create Reply Breakpoint	Set for a reply (inbound) interaction. This is useful for scenarios in which only the reply is important.
Initiate WS Debugging	Initiate a debugging session. For example, the debugging session encompasses an initiating SOAP request from a web service to a BPEL process to an adapter reference binding component.

Red icons representing your interaction choice are added.

For example, if you select **Create Breakpoints Pair**, request and reply breakpoint icons are added. Figure 49-6 provides details.

Figure 49-6 Request and Reply Breakpoint Icons on a Service Binding Component



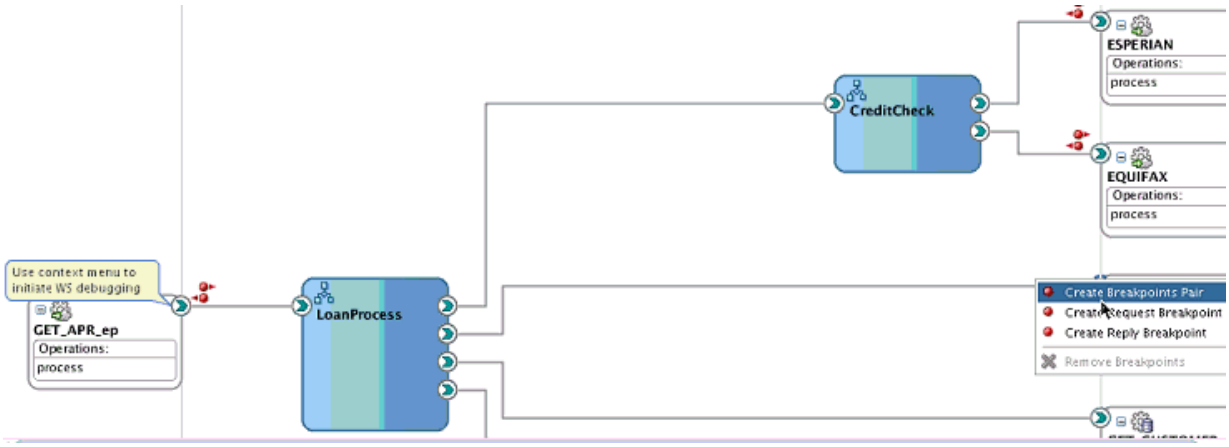
c. Go to Step 5.

3. To set a breakpoint on a reference binding component.

   a. Right-click the applicable reference binding component (for example, a web service or a database adapter), and select one of the breakpoint options described in Table 49-4.

      For example, if you select **Create Breakpoints Pair** for several references, request and reply breakpoint icons are added. Figure 49-7 provides details.

Figure 49-7 Breakpoints Set on Reference Binding Components

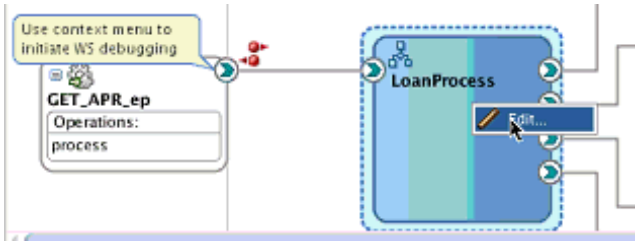


b. Go to Step 5.

4. To set a breakpoint on a service component (for this example, a BPEL process is selected).

   a. Select **Edit**, as shown in Figure 49-8.

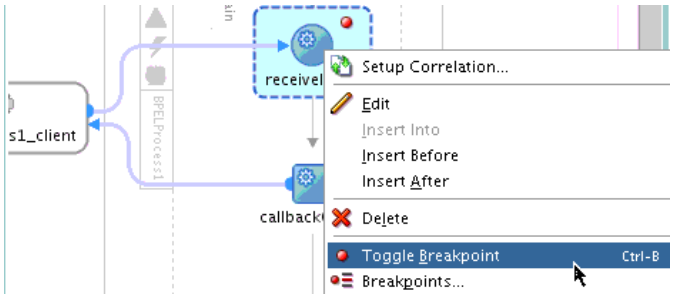
   Figure 49-8 Request and Reply Breakpoint Icons on a BPEL Process

   This opens the BPEL process in Oracle BPEL Designer.

   b. Right-click the BPEL activity on which to set a breakpoint, and select **Toggle Breakpoint**. Figure 49-9 provides details.

   Figure 49-9 Breakpoint Setting for a BPEL Process

   An icon is added to the activity. These breakpoint icons are only red dots because the flow is always in one direction. It is recommended that you always set a breakpoint on the first activity within an asynchronous BPEL process.

   c. To disable the breakpoint, right-click and select **Toggle Breakpoint** again. The red dot is removed. To display a list of all breakpoints set in the BPEL process, right-click the activity and select **Breakpoints**. You can also enable and disable breakpoints from this dialog.

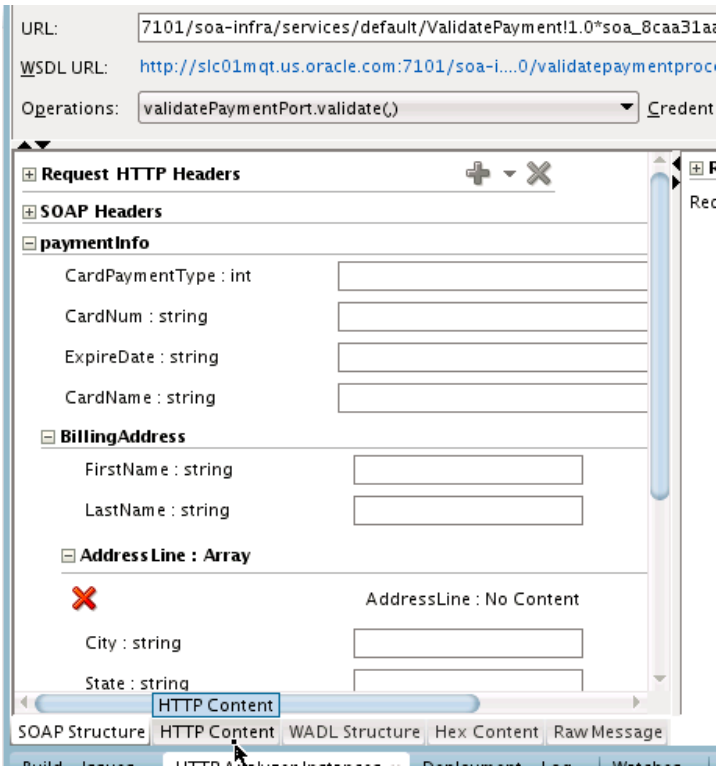
   d. Go to Step 5.

5. To begin debugging of the SOA composite application, right-click the right handle of the service binding component shown in Figure 49-5, and select **Initiate WS Debugging** from the menu.

   This invokes the HTTP Analyzer.

6. Enter the request message data to send, and click **Send Request** or click **HTTP Content** to copy and paste the contents from an XML file. You can either enter data field-by-field or copy and paste an XML document. Figure 49-10 provides details.

Figure 49-10 SOA Debugger Message Data

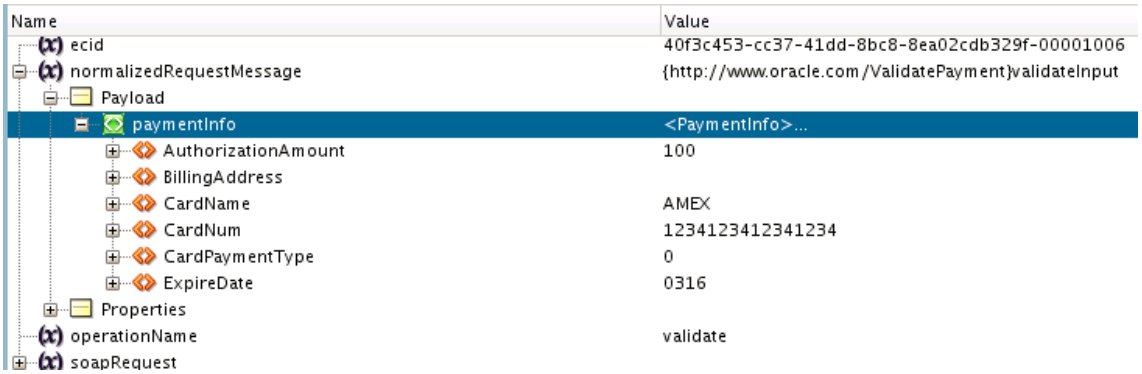


The debugger stops at the first breakpoint you set (for this example, on the service binding component).

7. In the Log window at the bottom of Oracle JDeveloper, click **Data**.

8. Expand the message contents. Figure 49-11 provides details. You can double-click a value to change it. For non-XML variables, right-click and choose **View value** (for example, the return message from a database adapter).

Figure 49-11 Message Contents After Debugger Invocation



How to Step Through a Debugging Session

When you create a breakpoint, a corresponding frame is created in the Structure window, as shown in Figure 49-12. This frame was created for the request-reply entry point on the service binding component.

A frame is a location. A stack of frames is a bread crumb trail of the locations that lead you to your current location. This is equivalent to a stack trace. It shows you where you are and how you got there. Frames are created independent of breakpoints. When you stop at a breakpoint, all frames that have been created in the Structure window are displayed. A stack frame also contains the data that existed at that point of time. Clicking a different stack frame in the Structure pane also updates the **Data** tab.

For example, if you have a web service connected to a BPEL process connected to a reference, if you set a breakpoint on the reference, you see a stack that generally looks as follows:

- Reference
- BPEL invoke
- BPEL scope
- Web service

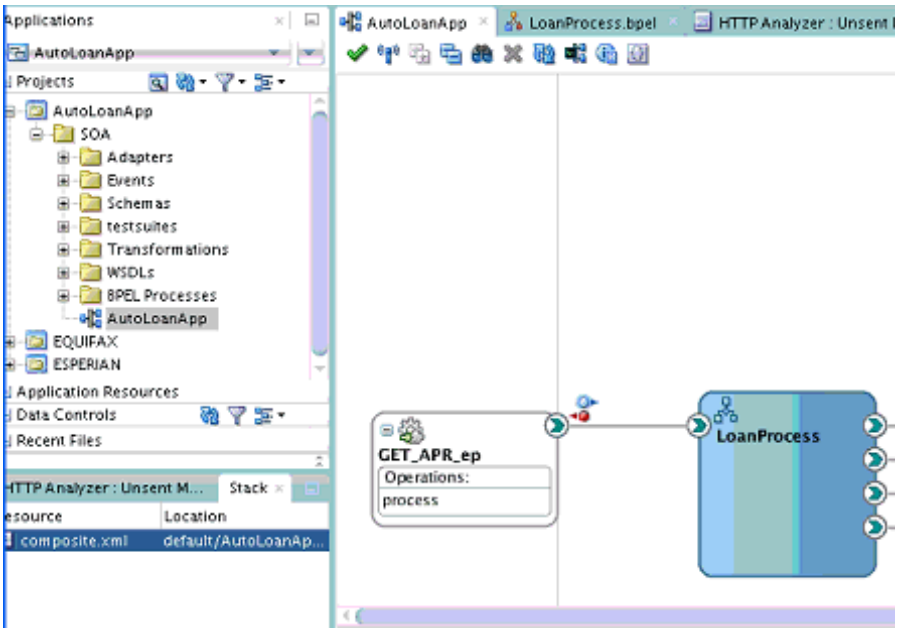
If you click the web service frame, the SOAP payload in the **Data** tab is displayed. If you then click the BPEL invoke frame, the various BPEL variables and other details are displayed in the **Data** tab.

You can step over the frame and begin debugging at a different location, such as a different breakpoint (for this example, the LoanProcess BPEL process). As you proceed with debugging, the following frames are created. Frames are where variables are located.

- Scope frame: Contains scope variables.
- Process frame: Contains global variables.

Variables are visible to a process from the top frame through the bottom frame. Frames are displayed in the Structure window.

Figure 49-12 Frames in Structure Window



To step through a debugging session:

1. Go to the tool bar in Oracle JDeveloper. The step options are shown in Figure 49-13.

Figure 49-13 Step Options in Oracle JDeveloper

Table 49-5 describes each option.

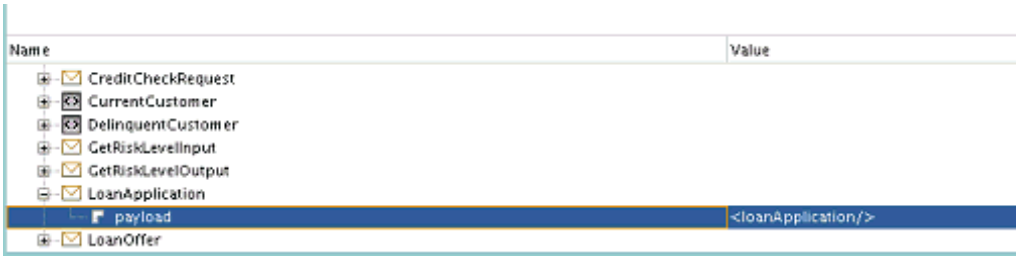
Table 49-5 Step Options in Oracle JDeveloper Main Menu

Icon	Description
	Ends or detaches from a debugging session.
	Steps over a frame. This places you at the next breakpoint (for example, the receive activity in the BPEL process on which a breakpoint was set in Figure 49-9). If there are no breakpoints, it steps over all the frames and goes back to the first frame. You can also press **F8** to step over a frame.
	Steps into the next valid location. This can be a new frame or the same frame, but in a different location. You can also press **F7** to step into a frame.
	Steps out of a frame. This option is only used to process a BPEL scope or sequence activity. After completion of scope processing, it pauses at the next scope or activity in the process. You can also press **Shift-F7**.
	Resumes a step operation. You can also press **F9** to resume.

2. If you selected the **Step Over** option, it stops at the receive activity.

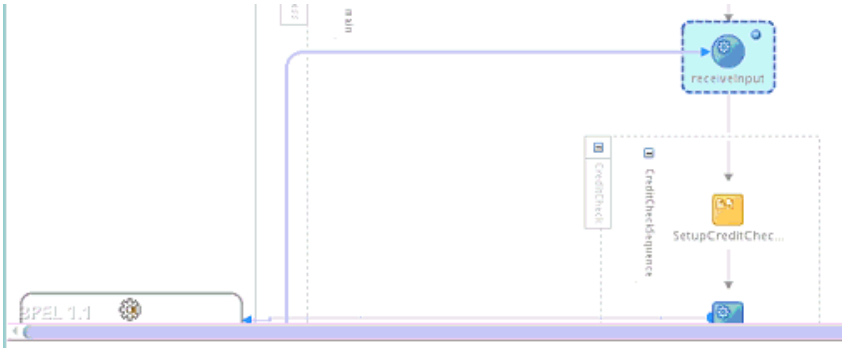
3. In the Log window, click **Data** and expand the contents to view the variables defined in the BPEL process, as shown in Figure 49-14. You can edit BPEL process variables during debugging. The payload is empty for the example shown in Figure 49-14.

Figure 49-14 Empty Payload



This is because the breakpoint on the receive activity has not been executed, as shown in Figure 49-15.

Figure 49-15 Empty Payload Before Receive Activity Breakpoint Execution



4. Click the **Step Into** option, as described in Table 49-5.

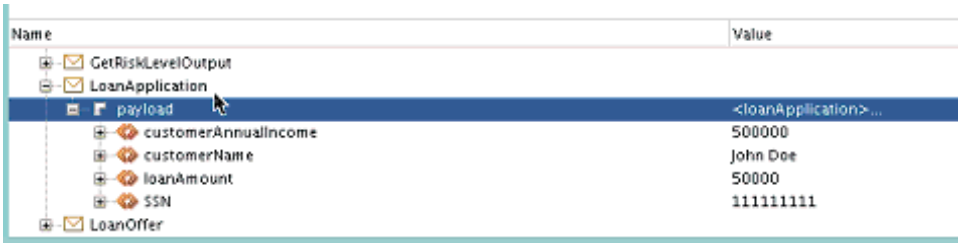
   This executes the receive activity shown in Figure 49-16.

Figure 49-16 Populated Payload After Receive Activity Breakpoint Execution

5. Expand the payload.

   The payload is populated with the data you entered in Step 6 of How to Set Breakpoints and Initiate Debugging. Figure 49-17 provides details.

Figure 49-17 Expanded Payload



6. Select the **Step Over** option, as described in Table 49-5. This causes the debugger to pause at the next breakpoint (for this example, a web service reference binding component, as shown in Figure 49-7).

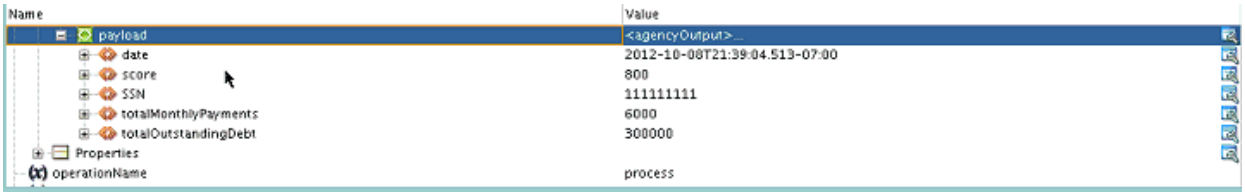
   The contents of the request message to the web service call are shown in Figure 49-18.

Figure 49-18 Request Message Payload Contents

7. Select the **Step Over** option.

8. Expand the payload to view the message reply. Figure 49-19 provides details.

Figure 49-19 Request Message Payload Contents



9. Proceed with debugging.

   If you step through the copy rules of an assign activity, the SOA debugger displays a window showing which copy rule it is on within the assign activity. The window has a table showing all the copy rules and there is a breakpoint icon next to the copy rule at which the debugger is stopped.

Note:

If you set a breakpoint on an adapter (for example, a database adapter), the SOA debugger steps out of the BPEL process service component and goes to the SOA Composite Editor.

How to End or Detach from a Debugging Session

To end or detach from a debugging session:

1. Click the button in the tools menu to end a debugging session. Figure 49-20 provides details.

Figure 49-20 End or Detach from a Debugging Session

The Terminate Debugger Process dialog is displayed.

2. Select an option. Table 49-6 provides details.

Table 49-6 Breakpoint Menu Options

Option	Description
Detach	Removes the debugger without ending the debugging process.
Terminate	Ends the debugging process.

3. If you selected **Detach**, click the debugger icon above the SOA Composite Editor shown in Figure 49-1 to resume debugging.
4. If you selected **Terminate**, right-click and select **Initiate WS Debugging** to reinitiate the debugger and start a new debugging session.

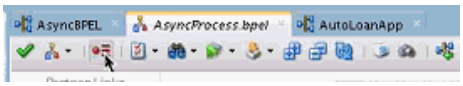
How to Remove Breakpoints

You can remove individual breakpoints or all breakpoints.

To remove breakpoints:

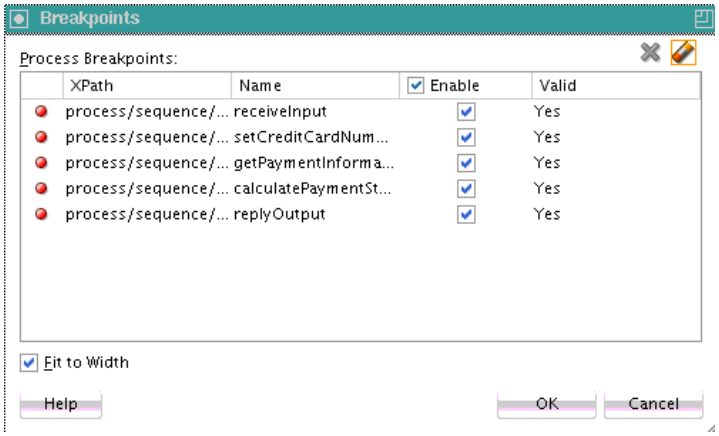
1. To remove an individual breakpoint, perform the following:
   - Right-click an activity on which a breakpoint has been set and select **Toggle Breakpoint**.
   - Click the **Breakpoints** icon above Oracle BPEL Designer and select the activity on which to remove a breakpoint in the Breakpoints dialog.
2. To remove all breakpoints, right-click in the SOA composite application, and select **Remove All Breakpoints**.
3. Click the icon above the BPEL process in Oracle BPEL Designer, as shown in Figure 49-21.

Figure 49-21 Breakpoints Icon in Oracle BPEL Designer



This invokes the Breakpoints dialog, as shown in Figure 49-22.

Figure 49-22 Breakpoints Dialog



4. In the **Enable** check boxes, select BPEL process breakpoints to disable.

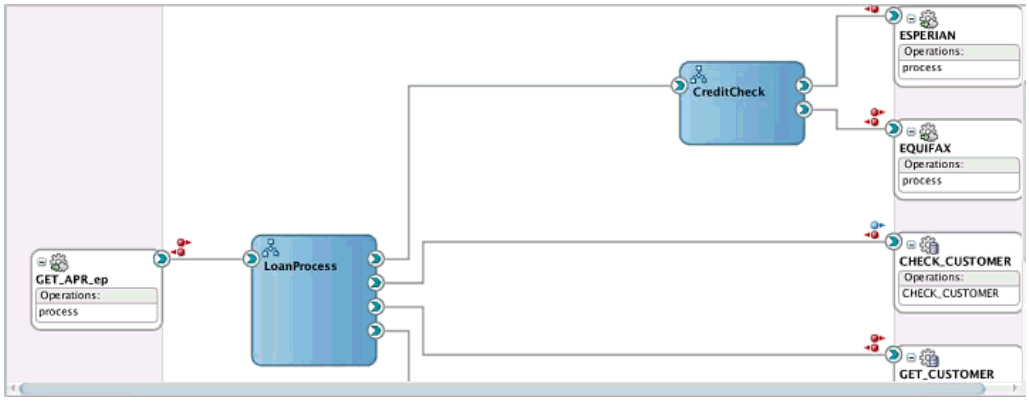
How to View Adapter Properties

You can view adapter properties under the **Data** tab in the Log window.

To view adapter properties:

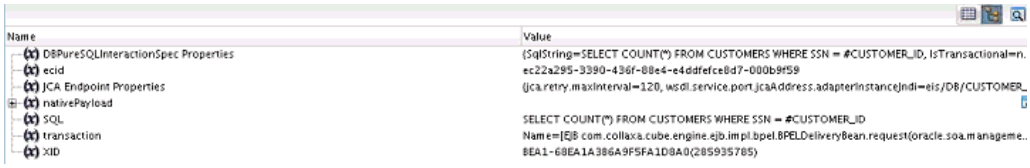
1. Click the **Step Over** icon until you stop at a breakpoint on a reference binding component such as a database adapter. Figure 49-23 provides details.

Figure 49-23 JCA Adapter Properties



The process is stopped to check on the existence of the customer. Adapter endpoint properties are displayed. Figure 49-24 provides details. The SQL syntax to be executed is also displayed.

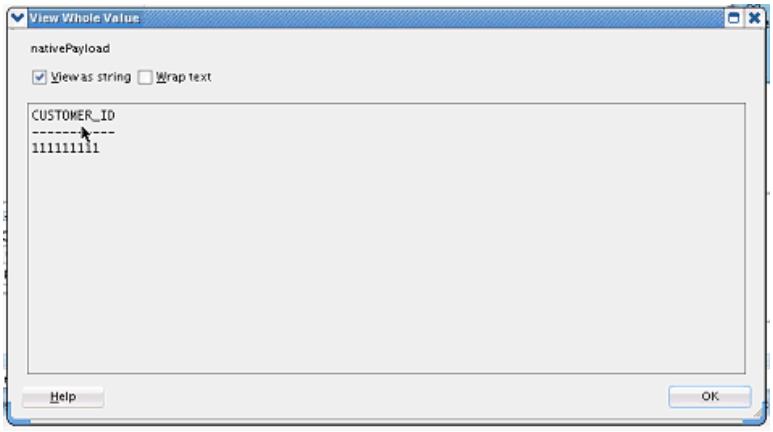
Figure 49-24 Adapter Output



2. Right-click a property and select **View Whole Value** to view the data being passed to the customer (for this example, **nativePayload** is selected). Figure 49-25 shows

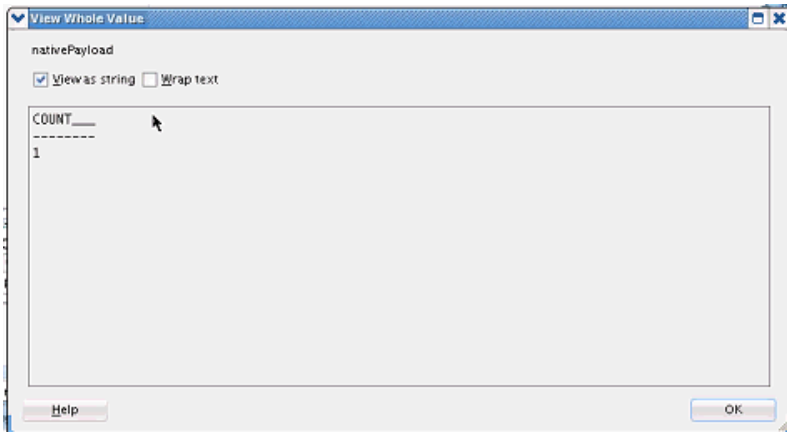
the customer ID being passed. **View Whole Value** is also useful for non-XML BPEL variables.

Figure 49-25 Request Message Contents Being Passed



3. Click the **Step Over** icon to execute the database adapter.
4. Right-click a property and select **View Whole Value** to view the customer reply message data. For this example, the value of **1** indicates that the customer exists. Figure 49-26 provides details.

Figure 49-26 Reply Message Contents Being Returned



5. To change a value, right-click a property and select **Modify Value**.

How to View Threads

A process instance is always run by a single logical thread, whether it is a synchronous or asynchronous process (the process ID can be thought of as the thread). The SOA debugger sees and uses the logical thread. If a process has a flow or flowN activity, then several logical threads run the flow or flowN activity.

To view threads:

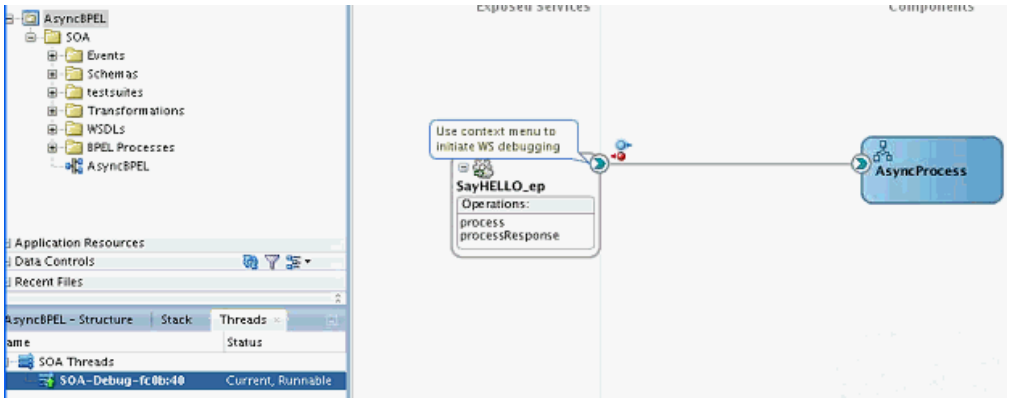
1. From the main menu, select **Window** > **Debugger** > **Threads**.

   The **Threads** tab is displayed in the Structure window.

2. Step into the service binding component of the BPEL process to begin debugging.

   The thread value for the request is **40**, as shown in the Structure window in Figure 49-27.

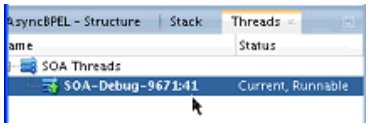
Figure 49-27 Request Thread Value



3. Step into the receive activity of the asynchronous BPEL process.

   The thread value for the reply is **41**, as shown in Figure 49-28.

Figure 49-28 Reply Thread Value



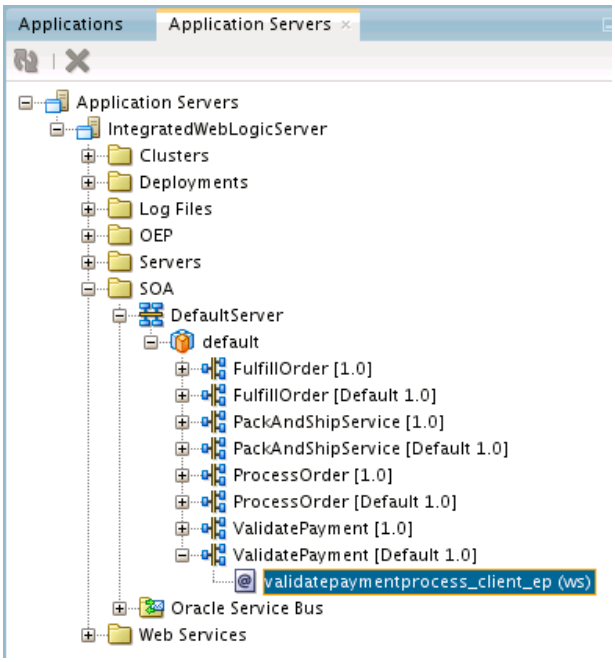
Testing SOA Composite Applications with the HTTP Analyzer

You can test HTTP requests and responses in a SOA composite application with the HTTP Analyzer in Oracle JDeveloper. The HTTP Analyzer enables you to examine the content of HTTP request/response package pairs. You can edit the content of a request package, resend it, and observe the response packet returned. For more information about the HTTP Analyzer, see the "Auditing and Monitoring Java Projects" chapter of *Developing Applications with Oracle JDeveloper*.

To test the SOA composite application with the HTTP Analyzer:

1. From the **Window** main menu, select **Application Servers**.
2. In the Application Servers window, expand the SOA composite application.
3. Right-click the component to test (for this example, a web service binding component), and select **Test Web Service**. Figure 49-29 provides details.

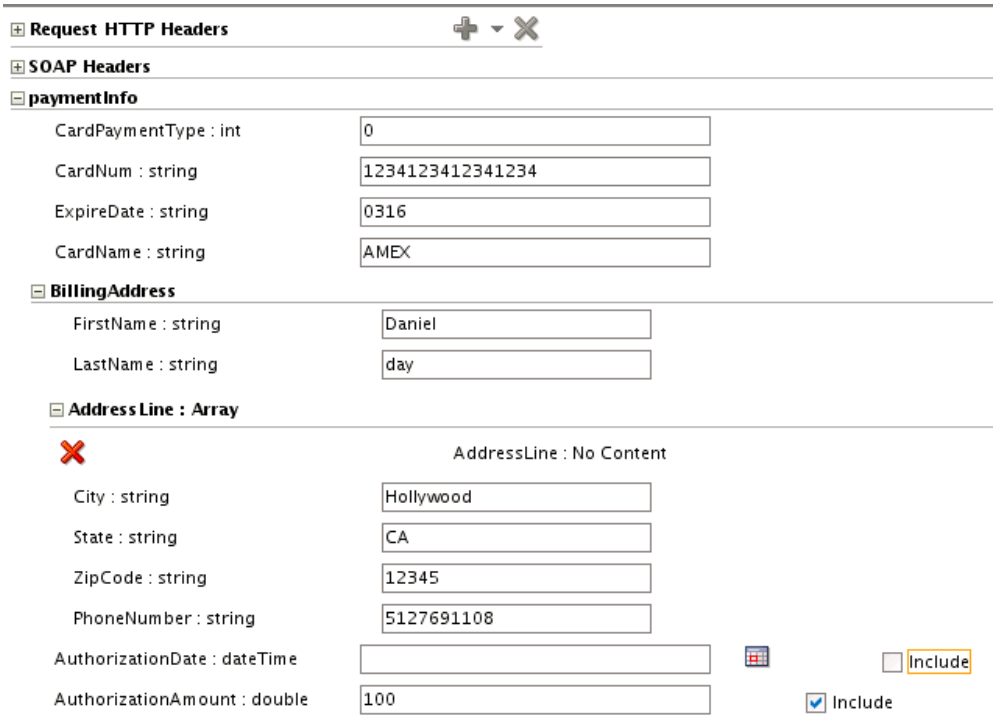
Figure 49-29 Component to Test in the Application Servers Window



The HTTP Analyzer is displayed.

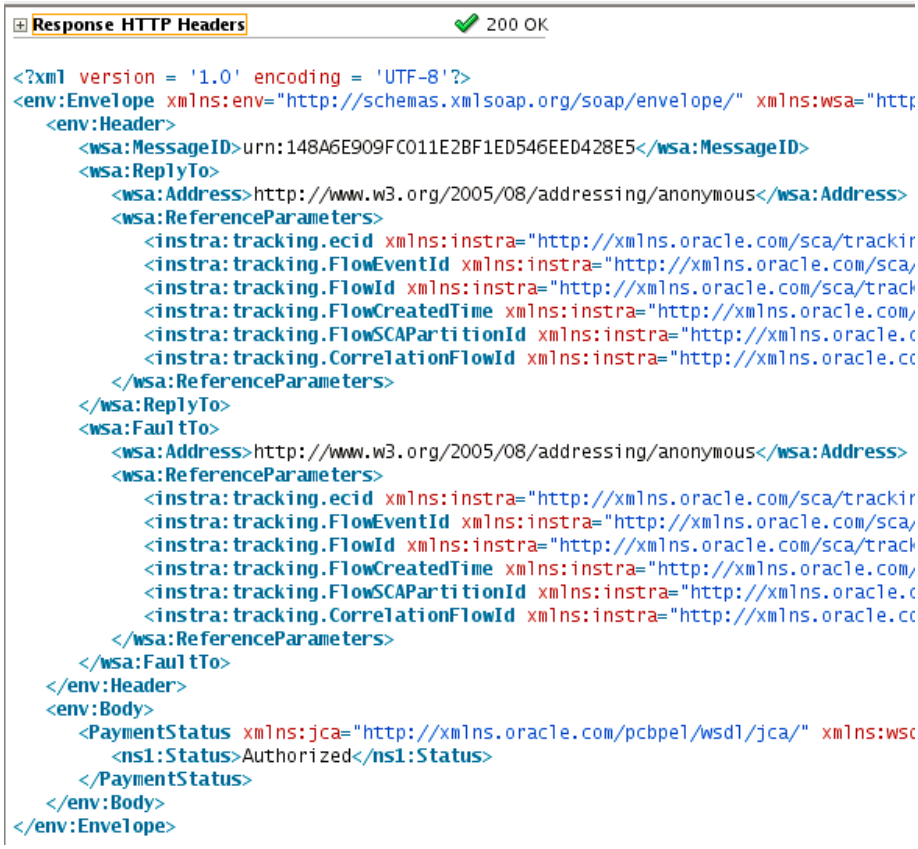
4. Enter the request message data to send, and click **Send Request** or click **HTTP Content** to copy and paste the contents from an XML file. Figure 49-30 provides details.

Figure 49-30 HTTP Analyzer



If successful, output similar to that shown in Figure 49-31 is displayed in the right pane.

Figure 49-31 Response HTTP Headers



You can also use the Test Web Service page to perform testing. For more information, see Section "Initiating a Test Instance of a Business Flow" of *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

Auditing SOA Composite Applications at the BPEL Activity Level

Audit trail data often accounts for a large percentage of the state data persisted to the database. To reduce the amount of persisted state data, you can specify finer-grained levels of auditing at the BPEL process activity level. These settings take precedence over the audit trail settings configured at the service component, SOA composite application, BPEL process service engine, and SOA Infrastructure levels.

You perform the following procedures:

- Create and configure an audit policy XML file that defines the level of auditing to perform on BPEL activities in the SOA composite application.
- Create and configure an audit policy binding XML file that binds the audit policy to the BPEL process.
- Place the files in the same directory location as the `composite.xml` file or in a separate directory that you identify with properties in the `composite.xml` file.
- Deploy the SOA composite application to the SOA Infrastructure.

- View the audit trail of the BPEL process activities in the flow trace of the SOA composite application in Oracle Enterprise Manager Fusion Middleware Control.

Note the following guidelines:

- The audit policy supports the auditing of both standard BPEL 1.1 and 2.0 activities and scopes and BPEL extension activities, such as emails, notifications, and all others. Within a parent scope, you can configure specific child scopes to be audited, and other child scopes to not be audited.
- The supported auditing levels are shown in Table 49-7.

Table 49-7 Auditing Levels

Level	Description
`Inherit`	Logging matches the SOA Infrastructure audit level that you set on the SOA Infrastructure Common Properties page in Oracle Enterprise Manager Fusion Middleware Control. This is the default setting.
`Production`	Minimal information for business flow instances is collected. For example, the BPEL process service engine does not capture the payload. Therefore, the payload details are not available in the flow audit trails. This level is optimal for most standard operations and testing.
`Development`	Complete information for BPEL process activities is collected. This option allows both composite instance tracking and payload tracking. This setting may have an impact on performance because the payload is stored at each step in the message flow. This setting is useful for debugging purposes.
`Off`	No logging is performed. Composite instance tracking information and payload tracking information are not collected.

- Support is provided for wild-card matching of process names and revision numbers in the fault policy binding file. For example:
  - Entering `Order*` applies to BPEL process service components included in the composite named `OrderProcess`, `OrderRejected`, and `OrderConfirmed`:

    ```
    <process auditPolicy="noLoops" name="Order*"/>
    ```

  - Entering `1*` applies to composite revisions `1.0`, `1.1`, and `1.2`:

    ```
    <process auditPolicy="noAssign" name="*" revision="1.*"/>
    ```

The following example shows the audit policy schema to use:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://schemas.oracle.com/bpel/auditpolicy"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:tns="http://schemas.oracle.com/bpel/auditpolicy"
   elementFormDefault="qualified">
   <!-- activity can have a type or a name as optional attribute.-->
   <!-- Audit rules apply to all activities if no specific type or name is -->
   <!-- provided -->
   <xs:complexType name="Activity">
        <xs:attribute name="type" type="xs:QName" use="optional"/>
        <xs:attribute name="name" type="tns:idType" use="optional"/>
        <xs:attribute name="auditLevel" type="tns:auditLevelType" use="required"/>
   </xs:complexType>
   <xs:simpleType name="idType">
        <xs:restriction base="xs:string">
```

```
            <xs:minLength value="1"/>
          </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="auditLevelType">
          <xs:restriction base="xs:string">
               <xs:enumeration value="off"/>
               <xs:enumeration value="minimal"/>
               <xs:enumeration value="production"/>
               <xs:enumeration value="development"/>
          </xs:restriction>
    </xs:simpleType>
    <xs:element name="auditPolicy">
         <xs:complexType>
              <xs:sequence>
                   <xs:element name="activity" type="tns:Activity" minOccurs="0"
                    maxOccurs="unbounded"/>
              </xs:sequence>
              <xs:attribute name="id" type="tns:idType" use="required"/>
              <xs:attribute name="version" type="xs:string" default="1.0"/>
         </xs:complexType>
         <!-- we restrict users to provide mulitple rules for same activity -->
         <xs:key name="UniqueActivity">
              <xs:selector xpath="tns:activity"/>
              <xs:field xpath="@type"/>
              <xs:field xpath="@name"/>
         </xs:key>
      </xs:element>
</xs:schema>
```

The following example shows the audit policy binding schema to use.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://schemas.oracle.com/bpel/auditpolicyBinding"
     xmlns:tns="http://schemas.oracle.com/bpel/auditpolicy"
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      elementFormDefault="qualified">
        <xs:complexType name="Process">
             <xs:attribute name="auditPolicyId" type="tns:idType" use="optional"/>
             <xs:attribute name="name" type="tns:idType" use="optional"/>
             <xs:attribute name="revision" type="tns:idType" use="optional"/>
        </xs:complexType>
        <xs:simpleType name="idType">
                <xs:restriction base="xs:string">
                       <xs:minLength value="1"/>
                </xs:restriction>
        </xs:simpleType>
        <xs:element name="auditPolicyBinding">
                <xs:complexType>
                       <xs:sequence>
                            <xs:element name="process" type="tns:Process"
                             minOccurs="0" maxOccurs="unbounded"/>
                        </xs:sequence>
                        <xs:attribute name="version" type="xs:string"
                         default="1.0"/>
                 </xs:complexType>
                 <xs:key name="UniqueActivity">
                       <xs:selector xpath="tns:process"/>
                       <xs:field xpath="@name"/>
                       <xs:field xpath="@revision"/>
                 </xs:key>
```

```
        </xs:element>
</xs:schema>
```

How to Audit SOA Composite Applications at the BPEL Activity Level

This section describes how to create and configure the audit policy and audit policy binding files.

To audit SOA composite applications at the BPEL activity level:

1. Create and configure an audit policy file (for example, named `audit-policy.xml`) that defines the audit level settings for the BPEL activities. The file can have any name and must follow the schema described in the preceding section.

```
<auditPolicies xmlns="http://schemas.oracle.com/bpel/auditpolicy"
xmlns:bpel="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:bpelx="http://schemas.oracle.com/bpel/extension" version="1.0">
  <auditPolicy name="whilePolicy">
     <!-- enabling this will cause all assign activities to not log -->
     <!-- anything to the audit trail -->
  <activity type="bpel:assign" auditLevel="production"/>

    <!-- enabling this will cause all scope activities and all -->
    <!-- enclosed activities to not log anything to the audit trail -->
     <activity type="bpel:scope" auditLevel="production"/>
     <!-- enabling this will cause all while activities to log with -->
     <!-- minimak level  -->
     <activity type="bpel:while" auditLevel="production"/>
      <activity type="bpel:reply" auditLevel="production"/>
      <activity type="bpel:flow" auditLevel="production"/>
       <activity type="bpel:switch" auditLevel="off"/>
       <activity type="bpel:terminate" auditLevel="production"/>
       <activity type="bpel:empty" auditLevel="development"/>
        <activity type="bpel:wait" auditLevel="production"/>
         <activity type="bpel:throw" auditLevel="off"/>
          <activity type="bpel:catchAll" auditLevel="production"/>
           <activity type="bpel:sequence" auditLevel="off"/>
            <activity type="bpel:receive" auditLevel="production"/>
  </auditPolicy>
</auditPolicies>
```

Note:

To enable BPEL extensions to be audited, enter `bpelx:exec` with an appropriate auditing level (for example, `production`).

```
<activity type="bpelx:exec" auditLevel="production"/>
```

---

2. Create and configure an audit policy binding XML file (for example, named `audit-binding.xml`) that binds the audit policy to the BPEL process. The file can have any name and must follow the schema described in the previous section. This example uses the wildcard option to enable all BPEL processes that begin with `myProcess` to be audited. Several other auditing options have been commented out.

```
<auditPolicyBindings xmlns="http://schemas.oracle.com/bpel/auditpolicyBinding"
 version="1.0">
  <!-- enabling this will cause all processes in the domain to use this -->
```

```
  <!-- policy audit -->
  <!-- <process auditPolicyName="whilePolicy" name="BPELProcess*"/> -->
  <!-- enabling this will cause all processes that start with the name -->
  <!-- myProcess to use the audit policy 'noLoops' -->
  <process auditPolicyName="noLoops" name="myProcess*"/>
  <!-- enabling this will cause all processes -->
  <!-- process auditPolicyName="noAssign" name="*"/> -->
</auditPolicyBindings>
```

3. Place the XML file in the same directory as the `composite.xml` file.
4. Define the `audit-policy.xml` and `audit-binding.xml` files in the `composite.xml` file.

```
<property name="oracle.composite.bpelAuditPolicyFile">audit-policy.xml</property>
<property
name="oracle.composite.bpelAuditBindingFile">audit-binding.xml</property>
```

5. Deploy the SOA composite application.

50

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. You can also create test cases for testing BPEL process service components included in the SOA composite application. 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 when it is ready for deployment to a production environment.

This chapter includes the following sections:

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

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 Creating Test Suites and Test Cases with the Create Composite Test Wizard.

Overview of Test Suites

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:

- Creating Test Suites and Test Cases with the Create Composite Test Wizard
- Editing the Contents of Test Cases in Test Mode in the SOA Composite Editor

Overview of Emulations

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:

- Emulations
- Editing the Contents of Test Cases in Test Mode in the SOA Composite Editor

Overview of Assertions

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 Assertions.

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.

Process Initiation

You first define the operation of your process in a binding component service such as a SOAP web service. The following example defines the operation of `initiate` to initiate the `TestFwk` SOA composite application. The initiation payload is also defined in this section:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- Generated by Oracle SCA Test Modeler version 1.0 at [6/13/07 10:50 AM]. -->
<compositeTest compositeDN="TestFwk"
 xmlns="http://xmlns.oracle.com/sca/2006/test">
  <about></about>
  <initiate serviceName="client" operation="initiate" isAsync="true">
    <message>
      <part partName="payload">
        <content>
          <loanApplication xmlns="http://www.autoloan.com/ns/autoloan">
            <SSN>111222333</SSN>
            <email>joe.smith@example.com</email>
            <customerName>Joe Smith</customerName>
            <loanAmount>20000</loanAmount>
            <carModel>Camry</carModel>
            <carYear>2007</carYear>
            <creditRating>800</creditRating>
          </loanApplication>
        </content>
      </part>
    </message>
  </initiate>
</compositeTest>
```

Emulations

You create emulations to simulate the message data that your SOA composite application receives from web service partners.

In the test code in the following example, the loan request is initiated with an error. A fault message is received in return from a web service partner:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- Generated by Oracle SCA Test Modeler version 1.0 at [7/3/07 3:29 PM]. -->
<compositeTest compositeDN="CompositeTest"
 xmlns="http://xmlns.oracle.com/sca/2006/test">
  <about></about>
  <initiate serviceName="client" operation="initiate" isAsync="true">
    <message>
      <part partName="payload">
        <filePath>loanApplication.xml</filePath>
      </part>
    </message>
  </initiate>
  <wireActions wireSource="LoanBroker/CreditRatingService" operation="process">
```

```
    <emulate duration="PT0S">
      <fault faultName="ser:NegativeCredit" xmlns:ser="http://services.otn.com">
        <message>
          <part partName="payload">
            <filePath>creditRatingFault.xml</filePath>
          </part>
        </message>
      </fault>
    </emulate>
  </wireActions>
</compositeTest>
```

Two message files, `loanApplication.xml` and `creditRatingFault.xml`, are invoked in this emulation. If the loan application request in `loanApplication.xml` contains a social security number beginning with `0`, the `creditRatingFault.xml` file returns the fault message shown in the following example:

```
<error xmlns="http://services.otn.com">
  Invalid SSN, SSN cannot start with digit '0'.
</error>
```

For more information, see Editing the Contents of Test Cases in Test Mode in the SOA Composite Editor.

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. The following example instructs Oracle SOA Suite to ensure that the content of the `customerName` variable matches the content specified.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- Generated by Oracle SCA Test Modeler version 1.0 at [6/13/07 10:51 AM]. -->
<compositeTest compositeDN="TestFwk"
 xmlns="http://xmlns.oracle.com/sca/2006/test">
  <about></about>
  <initiate serviceName="client" operation="initiate" isAsync="true">
    <message>
      <part partName="payload">
        <content>
          <loanApplication xmlns="http://www.autoloan.com/ns/autoloan">
            <SSN>111222333</SSN>
            <email>joe.smith@example.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 Editing the Contents of Test Cases in Test Mode in the SOA Composite Editor.

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 Editing the Contents of Test Cases in Test Mode in the SOA Composite Editor.

Creating Test Suites and Test Cases with the Create Composite Test Wizard

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.

You can create test suites and test cases in either of two ways:

- In the Applications window
- From the Oracle JDeveloper main menu

Both options invoke the Create Composite Test wizard, which enables you to define the initiating operation, callback operation, and input and output messages.

Note:

Do *not* enter a multibyte character string as a test suite name or test case name. Doing so causes an error to occur when the test is executed from Oracle Enterprise Manager Fusion Middleware Control.

---

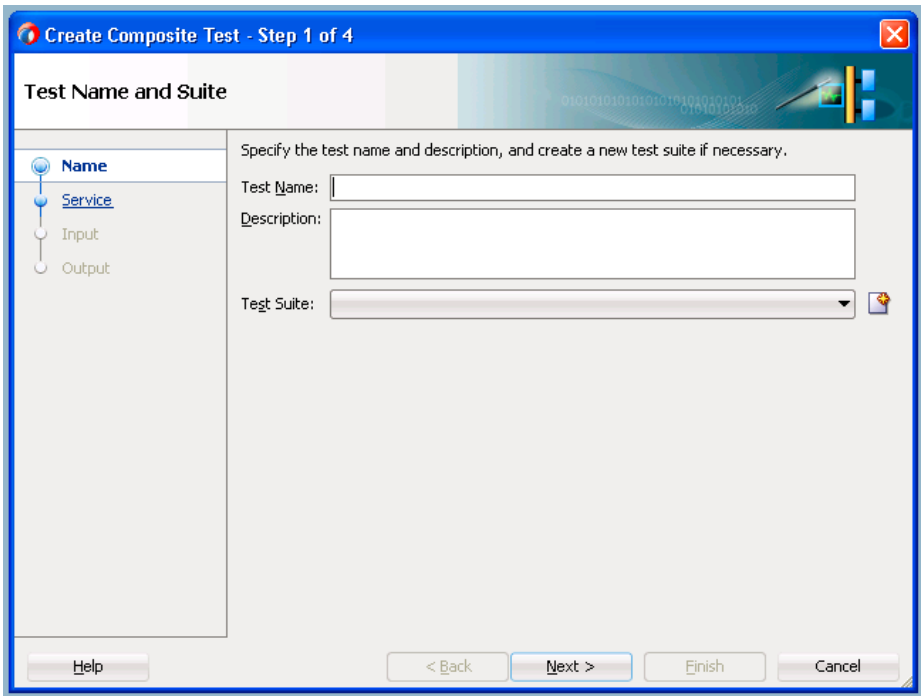
1. Perform one of the following steps to create a new test suite or create a new composite test in an existing test suite. Table 50-1 provides details.

Table 50-1 Test Suite Creation or Selection

From the...	Perform...
Oracle JDeveloper main menu	**a.** Select **File** > **New** > **Application** > **SOA Tier** > **Tests** > **Composite Test Suite**. The Create Test Suite dialog is displayed. **b.** Enter a test suite name, and click **OK**. or **a.** Select **File** > **From Gallery** > **SOA Tier** > **Tests** > **Composite Test Suite**. The Create Test Suite dialog is displayed. **b.** Enter a test suite name, and click **OK**.
Applications window	**a.** Right-click the **testsuites** folder and select **Create Test Suite**. The Create Test Suite dialog is displayed. **b.** Enter a test suite name, and click **OK**.
Structure window	**a.** Right-click **Test Suites** and select **Create Test Suite**. The Create Test Suite dialog is displayed. **b.** Enter a test suite name, and click **OK**.
Oracle JDeveloper main menu	**a.** Select **File** > **New** > **Application** > **SOA Tier** > **Tests** > **Composite Test**. or **a.** Select **File** > **New** > **Composite Test**. **Note**: Both selections provide the option of creating a new test suite or selecting an existing test suite in which to include the new composite test.

The Create Composite Test Wizard - Test Name and Suite page appears, as shown in Figure 50-1.

Figure 50-1 Create Composite Test Wizard - Test Name and Suite Page



This wizard enables you to create simple tests without manually creating test details in test mode in the SOA Composite Editor, as described in Editing the Contents of Test Cases in Test Mode in the SOA Composite Editor. You only must manually use this editor in test mode if you want to add additional test metadata such as emulations.

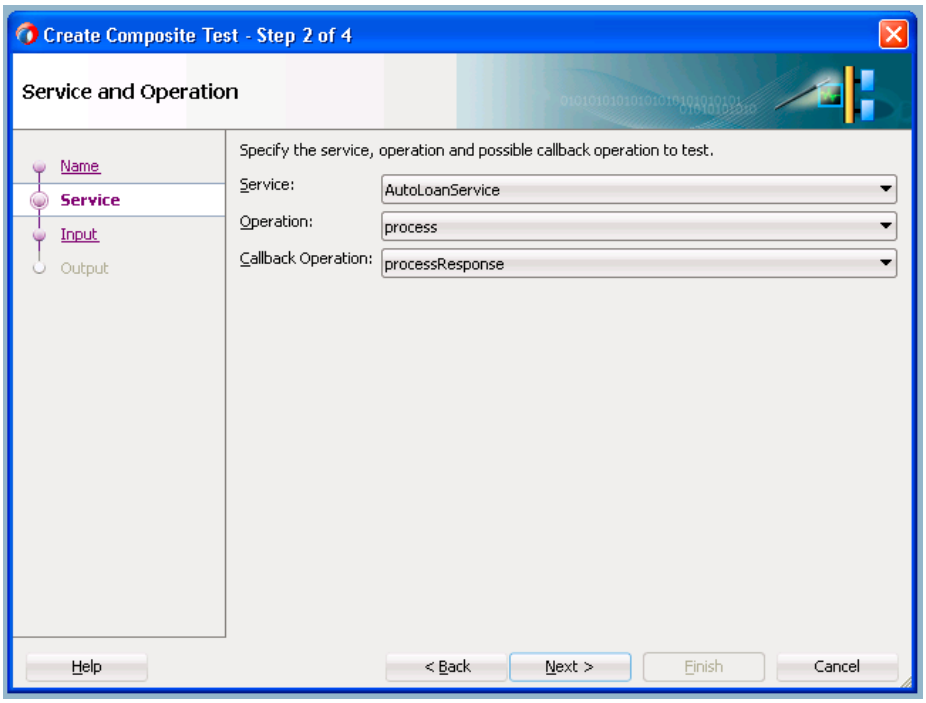
2. Provide values appropriate to your environment, as described in Table 50-2, and click **Next**.

Table 50-2 Create Composite Test Wizard - Test Name and Suite Page

Field	Description
Test Name	Enter a name for the test.
Description	Enter an optional description of the test. The description is displayed in the **Description** column of the Test Cases page of the **Unit Tests** tab in Oracle Enterprise Manager Fusion Middleware Control.
Test Suite	Select an existing test suite to include this test or click the icon to create a new test suite in the Create Test Suite dialog.

The Create Composite Test Wizard - Service and Operation page appears, as shown in Figure 50-2.

Figure 50-2 Create Composite Test Wizard - Service and Operation Page



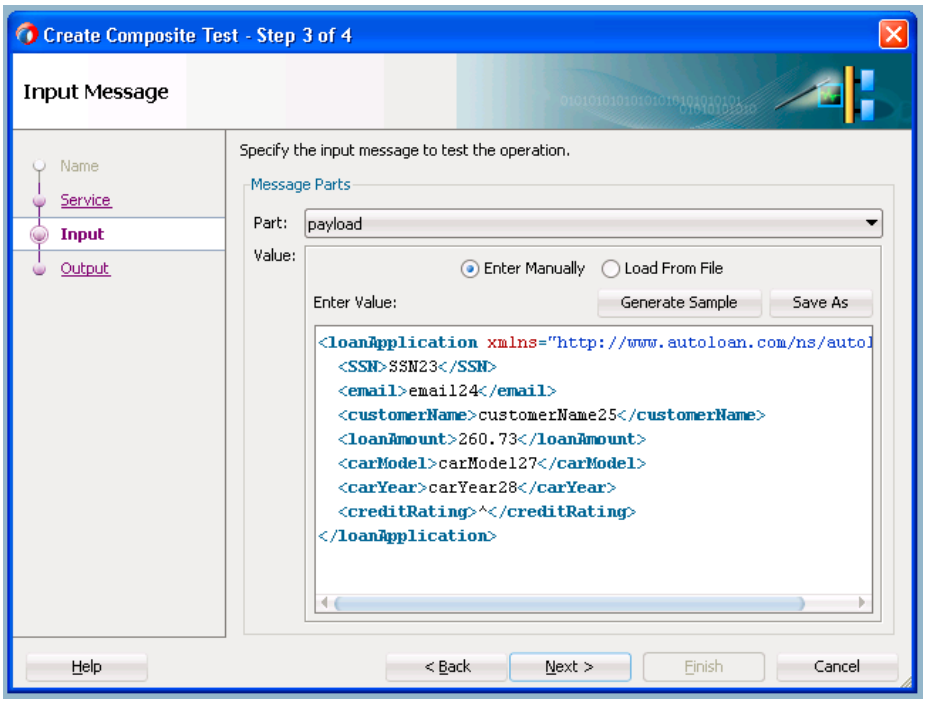
3. Provide values appropriate to your environment, as described in Table 50-3, and click **Next**.

Table 50-3 Create Composite Test Wizard - Service and Operation Page

Field	Description
Service	Select the SOA composite application to test.
Operator	Select the operation.
Callback Operation	Optionally select the callback (response) operation.

The Create Composite Test Wizard - Input Message page appears, as shown in Figure 50-3. This page enables you to specify the input message to test the operation.

Figure 50-3 Create Composite Test Wizard - Input Message Page



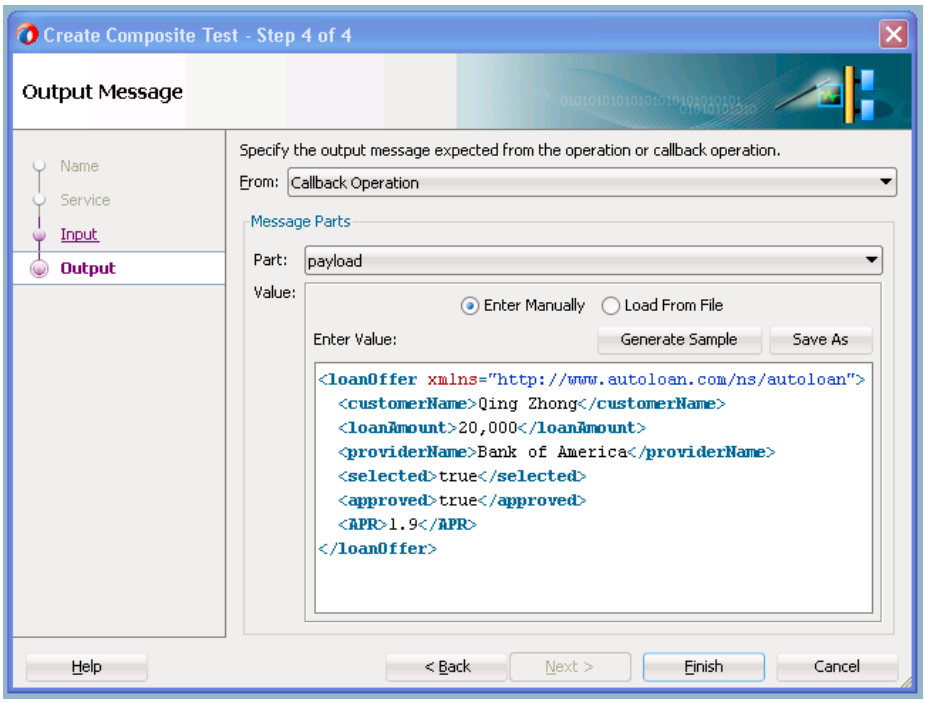
Provide values appropriate to your environment, as described in Table 50-4, and click **Next**.

Table 50-4 Create Composite Test Wizard - Input Message Page

Field	Description
Part	Select the message part containing the input (for example, **payload**). If the operation input message has multiple parts, then specify each message part by changing the part name, one by one. For each message part, you can either enter the XML document contents manually or you can load the document from an XML file.
Value	Create a simulated input message to send to 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 from the message part schema for testing. Click **Save As** to save the sample file for later use by the same test or other tests in the same test suite.
• **Load From File**	Click the **Browse** icon to load message data from a file. The file is added to the **messages** folder in the Applications window.

The Create Composite Test Wizard - Output Message page appears, as shown in Figure 50-4. This page specifies the output message expected from the operation or callback operation.

Figure 50-4 Create Composite Test Wizard - Output Message Page



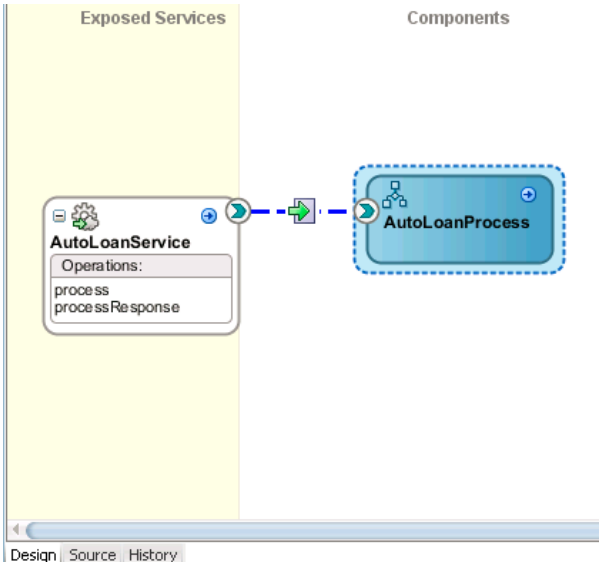
Provide values appropriate to your environment, as described in Table 50-5, and click **Finish**.

Table 50-5 Create Composite Test Wizard - Output Message Page

Field	Description
From	Select the external web service from which to receive the message.
Part	Select the message part containing the output (for example, **payload**). If the operation input message has multiple parts, then specify each message part by changing the part name, one by one. For each message part, you can either enter the XML document contents manually or you can load the document from an XML file.
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 Applications window.

The test suite is created, and the test mode of the SOA Composite Editor is displayed to show the test. Figure 50-5 provides details. You can add additional test metadata such as emulations, if necessary. If the current test is complete, you can continue to create another test by clicking the test image button on the toolbar. If you want to run the test, you can press the green arrow button.

Figure 50-5 Test Suite Creation



A test is created in the Applications window, along with the following subfolders:

- **componenttests**
- **includes**
- **messages**

  Contains message test files that you load into this directory through the test mode user interface.

- **tests**

  Contains the XML file for the test suite.

A folder named after the test suite also displays in the Structure window. This indicates that you are in the test mode of the . 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.

The following operating system test suite directory is also created:

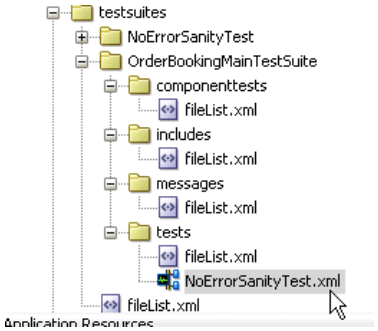
`C:\JDeveloper\mywork\application_name\project_name\testsuites\test_suite_name`

4. If you want to exit test mode and return to design mode in the , click the last icon above the designer. Figure 50-6 provides details.

Figure 50-6 Test Mode Exit

5. Save your changes when prompted.
6. Under the **testsuites** folder in the Applications window, double-click the XML file name to return to test mode. Figure 50-7 provides details.

Figure 50-7 Test Mode Access



Note:

- 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 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.

Editing the Contents of Test Cases in Test Mode in the SOA Composite Editor

After creating the basic contents of test suites and test cases with the Create Composite Test Wizard, you can make additional updates in the test mode of the SOA Composite Editor.

Test cases consist of process initiations, emulations, and assertions. 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.

Note:

You can also edit test case contents in the Property Inspector. Single-click the component or wire to edit to invoke the Property Inspector at the bottom of the page for editing.

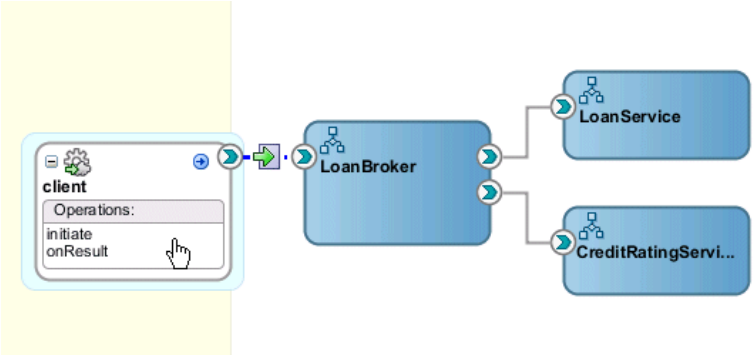
How to Initiate Inbound Messages

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 50-8.

Figure 50-8 Service Binding Component Access



The Initiate Messages dialog appears.

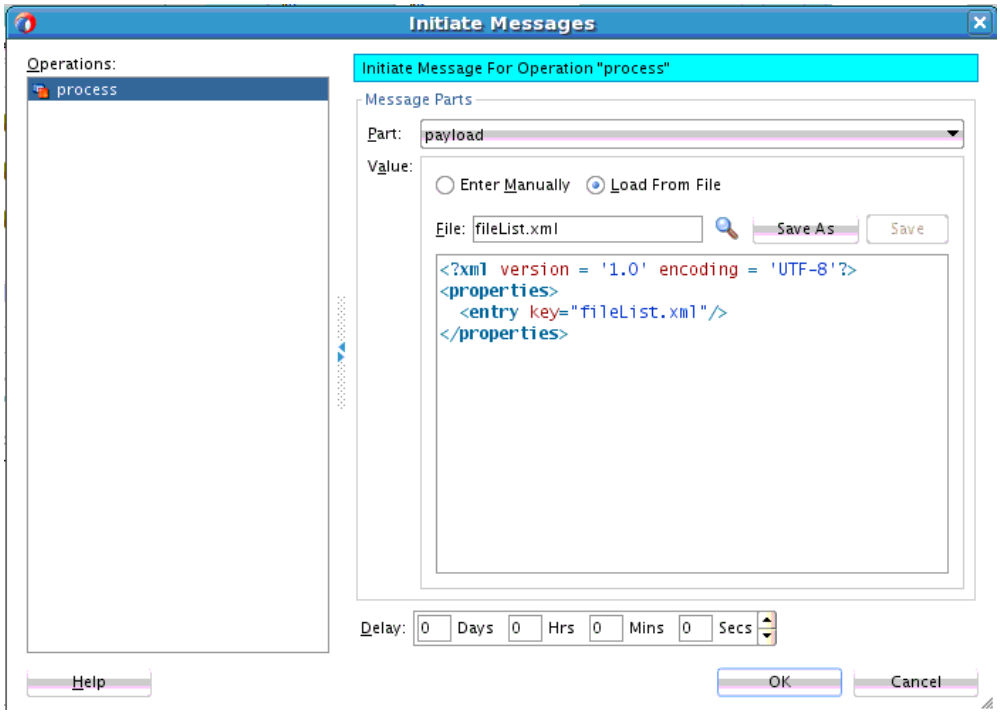
3. Enter the details shown in Table 50-6:

Table 50-6 Initiate Messages 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 Applications window.

Figure 50-9 shows this dialog:

Figure 50-9 Initiate Messages Dialog



The inbound process initiation message from a client looks as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- Generated by Oracle SCA Test Modeler version 1.0 at [7/12/07 8:36 AM]. -->
<compositeTest compositeDN="CompositeTest"
 xmlns="http://xmlns.oracle.com/sca/2006/test">
  <about/>
  <initiate serviceName="client" operation="initiate" isAsync="true">
    <message>
      <part partName="payload">
        <filePath>loanApplication.xml</filePath>
      </part>
    </message>
  </initiate>
. . .
. . .
```

The `loanApplication.xml` referenced in the process initiation file contains a loan application payload:

```
<loanApplication xmlns="http://www.autoloan.com/ns/autoloan">
  <SSN>111222333</SSN>
  <email>joe.smith@example.com</email>
  <customerName>Joe Smith</customerName>
  <loanAmount>20000</loanAmount>
  <carModel>Camry</carModel>
  <carYear>2007</carYear>
  <creditRating>800</creditRating>
</loanApplication>
```

4. Click **OK**.

How to Emulate Outbound Messages

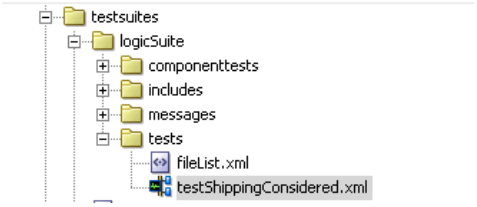
To emulate outbound messages:

> **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 Applications window, double-click a test case. Figure 50-10 provides details.

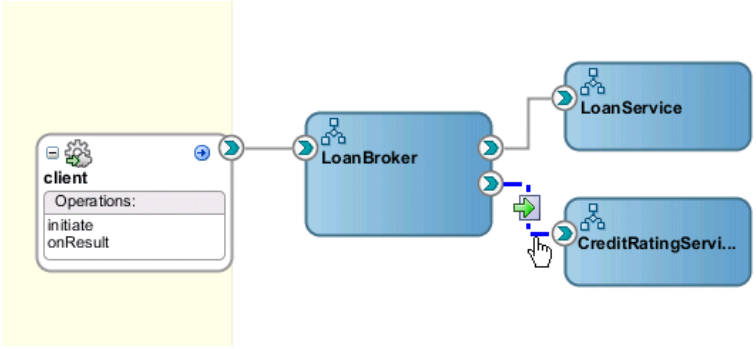
Figure 50-10 Test Case Access



The SOA composite application in the is refreshed to display in test mode. This mode enables you to define test information.

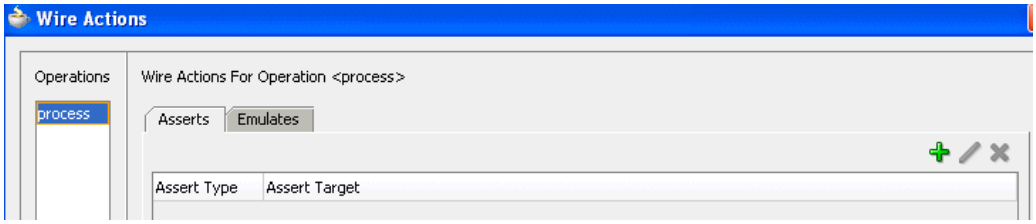
3. Double-click the wire of the SOA composite application area to test. For the example shown in Figure 50-11, the wire between the **LoanBroker** process and the synchronous **CreditRating** web service is selected.

Figure 50-11 Wire Access



This displays the Wire Actions dialog shown in Figure 50-12, from which you can design emulations and assertions for the selected part of the SOA composite application.

Figure 50-12 Wire Actions Dialog



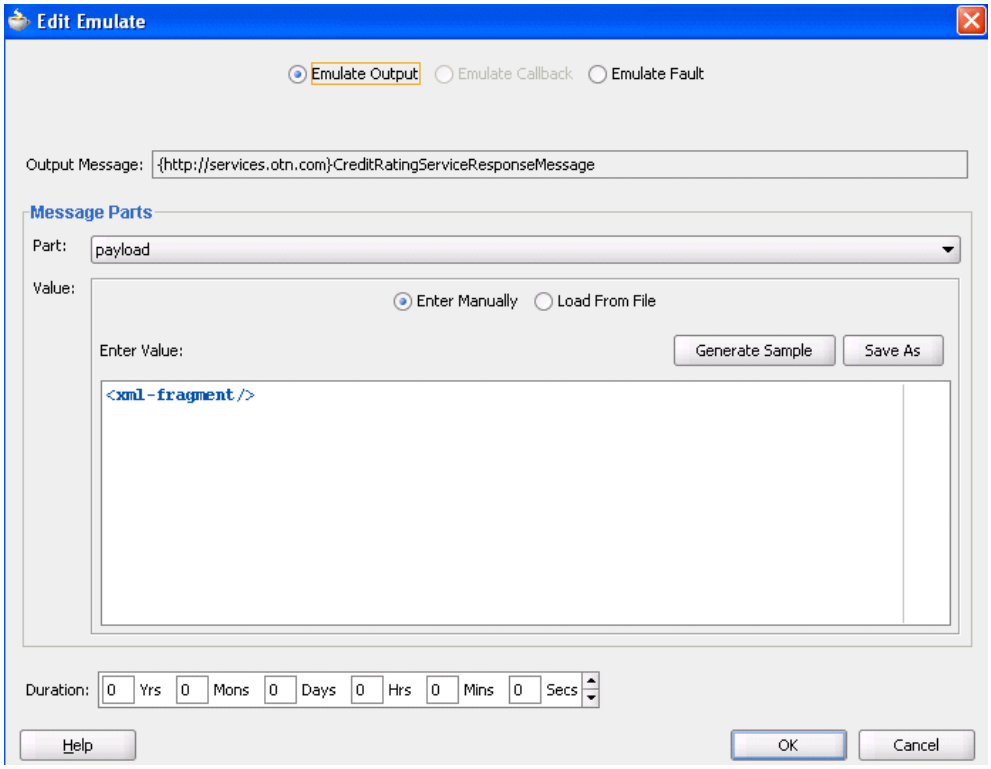
4. Click the **Emulates** tab.
5. Click the **Add** icon.
6. Click **Emulate Output**.
7. Enter the details described in Table 50-7:

Table 50-7 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 Applications window.
Duration	Enter the maximum amount of time to wait for the message to be delivered from the web service partner.

Figure 50-13 shows this dialog:

Figure 50-13 Emulate Dialog with Emulate Output Selected



A simulated output message from a synchronous web service partner that you enter manually or load from a file looks as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- Generated by Oracle SCA Test Modeler version 1.0 at [7/3/07 3:26 PM]. -->
<compositeTest compositeDN="CompositeTest"
 xmlns="http://xmlns.oracle.com/sca/2006/test">
  <about></about>
  <initiate serviceName="client" operation="initiate" isAsync="true">
    <message>
      <part partName="payload">
        <filePath>loanApplication.xml</filePath>
      </part>
    </message>
  </initiate>
  <wireActions wireSource="LoanBroker/CreditRatingService" operation="process">
    <emulate 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**.

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.

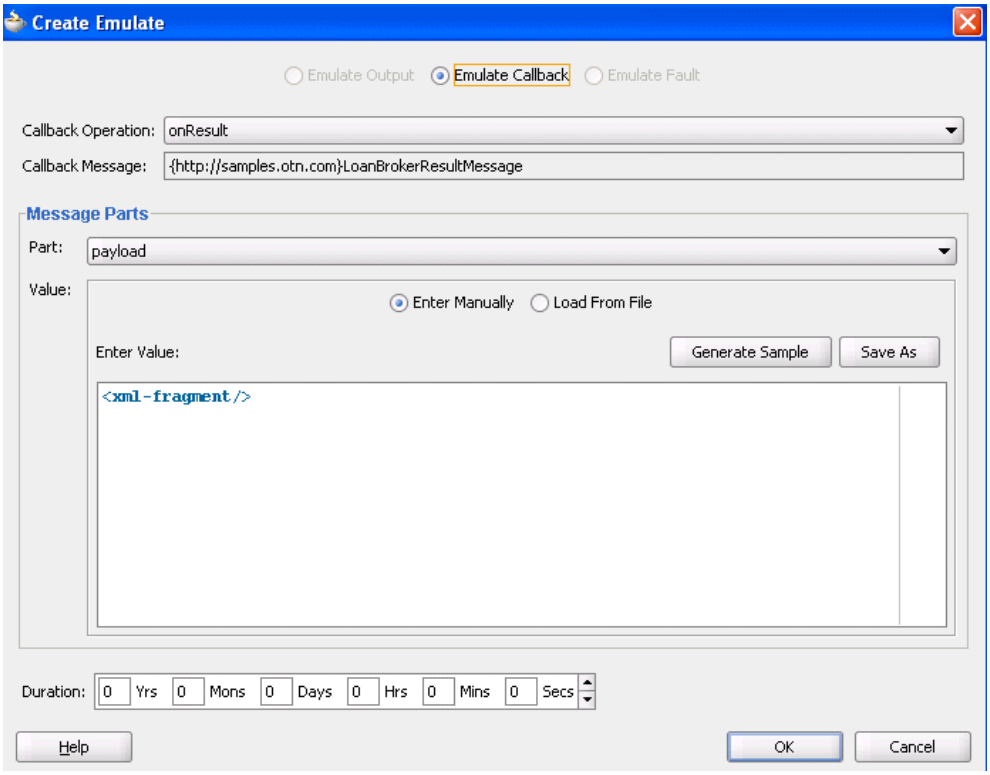
1. Access the Wire Actions dialog by following Step 1 through Step 3 of How to Emulate Outbound Messages.
2. Click the **Emulates** tab.
3. Click the **Add** icon.
4. Click **Emulate Callback**. This field is only enabled for asynchronous processes.
5. Enter the details described in Table 50-8:

Table 50-8 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 Applications window.
Duration	Enter the maximum amount of time to wait for the callback message to be delivered from the web service partner.

Figure 50-14 shows this dialog:

Figure 50-14 Emulate Dialog with Emulate Callback Selected



The simulated callback message from a web service partner looks as follows. You enter this message manually or load it from a file:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- Generated by Oracle SCA Test Modeler version 1.0 at [7/3/07 3:27 PM]. -->
<compositeTest compositeDN="CompositeTest"
 xmlns="http://xmlns.oracle.com/sca/2006/test">
  <about></about>
  <initiate serviceName="client" operation="initiate" isAsync="true">
    <message>
      <part partName="payload">
        <filePath>loanApplication.xml</filePath>
      </part>
    </message>
  </initiate>
  <wireActions wireSource="LoanBroker/LoanService" operation="initiate">
    <emulate callbackOperation="onResult" duration="PT0S">
      <message>
        <part partName="payload">
          <filePath>loanOffer.xml</filePath>
        </part>
      </message>
    </emulate>
  </wireActions>
</compositeTest>
```

The `loanOffer.xml` message file referenced in the callback message provides details about the credit rating approval.

```
<loanOffer xmlns="http://www.autoloan.com/ns/autoloan">
  <providerName>Bank Of America</providerName>
  <selected>false</selected>
```

```
  <approved>true</approved>
  <APR>1.9</APR>
</loanOffer>
```

6. Click **OK**.

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.

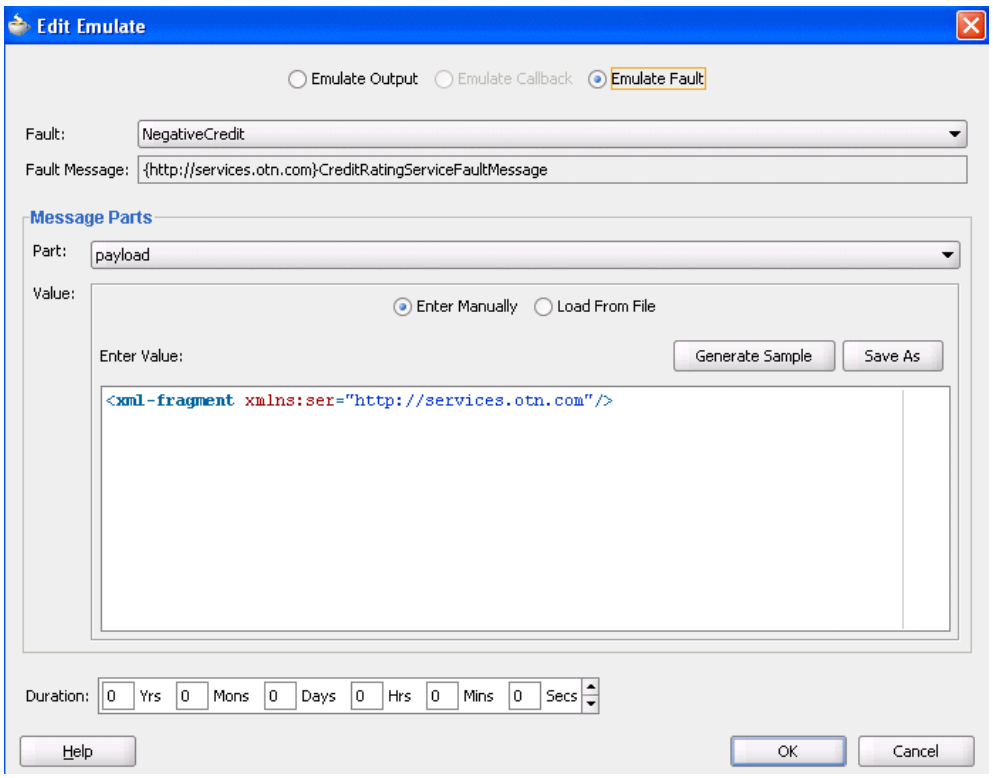
1. Access the Wire Actions dialog by following Step 1 through Step 3 of How to Emulate Outbound Messages.
2. Click the **Emulates** tab.
3. Click the **Add** icon.
4. Click **Emulate Fault**.
5. Enter the details described in Table 50-9:

Table 50-9 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 Applications window.
Duration	Enter the maximum amount of time to wait for the fault message to be delivered from the web service partner.

Figure 50-15 shows this dialog:

Figure 50-15 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 Emulations.

6. Click **OK**.

How to Create Assertions

To create assertions:

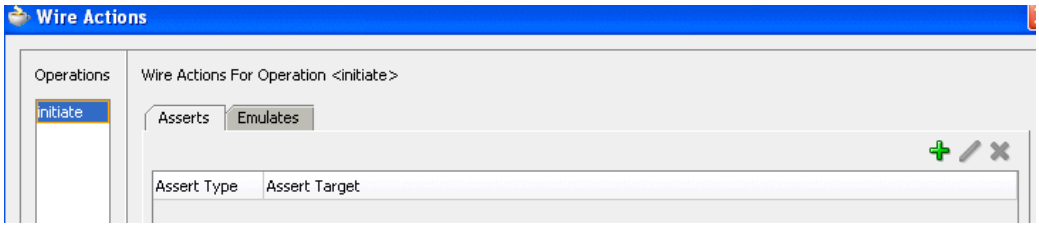
You perform assertions to verify variable data or process flow. Assertions enable you to validate test data in an entire XML document, a part section of a message, a nonleaf element, or a leaf element as a process is executed. This is done by extracting a value and comparing it to an expected value.

1. Access the Wire Actions dialog by following Step 1 through Step 3 of How to Emulate Outbound Messages.

2. Click the **Asserts** tab.

   Figure 50-16 shows this dialog:

Figure 50-16 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 50-10. 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 50-10 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 50-11 based on the type of assertion you want to perform.

Table 50-11 Assertion Types

For an Assertion on...	See...
• A part section of a document • A nonleaf element • An entire XML document	Creating Assertions on a Part Section_ Nonleaf Element_ or Entire XML Document
A leaf element	Creating Assertions on a Leaf Element

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.

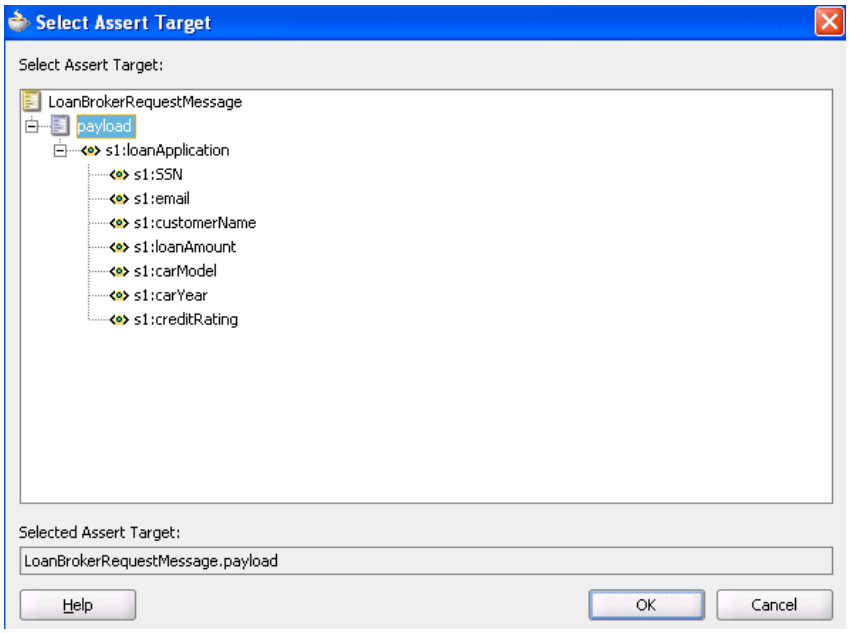
1. Click **Browse** to select the target part section, nonleaf element, or entire XML document to assert.

   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 50-17 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 50-17 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 50-12:

Table 50-12 Create Assert Dialog Fields and Values

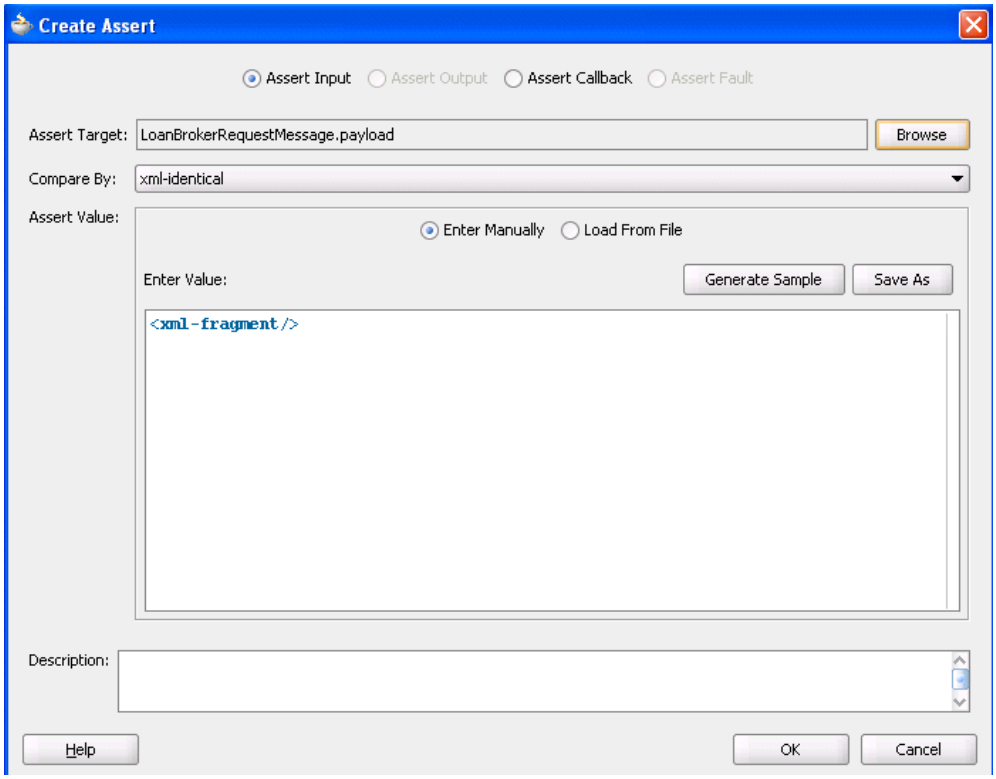
Field	Value
Fault	Select the type of fault to assert (for example, **NegativeCredit**). This field only displays if you select **Assert Fault** in Step 4.of How to Create Assertions.
Assert Target	Displays the assert target you selected in Step 2.

Table 50-12 (Cont.) Create Assert Dialog Fields and Values

Field	Value
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`
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 Applications window.
Description	Enter an optional description.

Figure 50-18 shows this dialog with **Assert Input** selected:

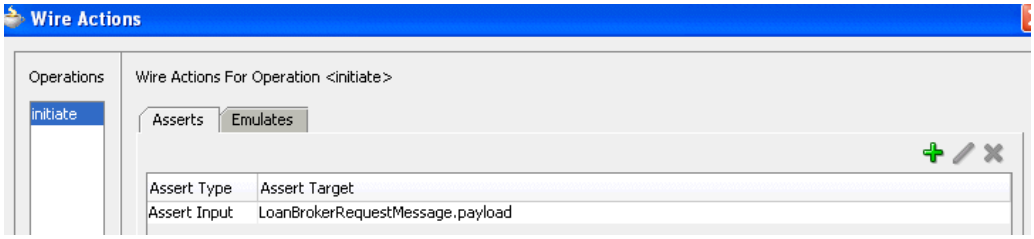
Figure 50-18 Create Assert Dialog with Assert Input Selected



4. Click **OK**.

   The Wire Actions dialog shown in Figure 50-19 displays your selection.

Figure 50-19 Wire Actions Dialog with Asserts Tab Selected



5. Click **OK**.

Creating Assertions on a Leaf Element

To create assertions on a leaf element:

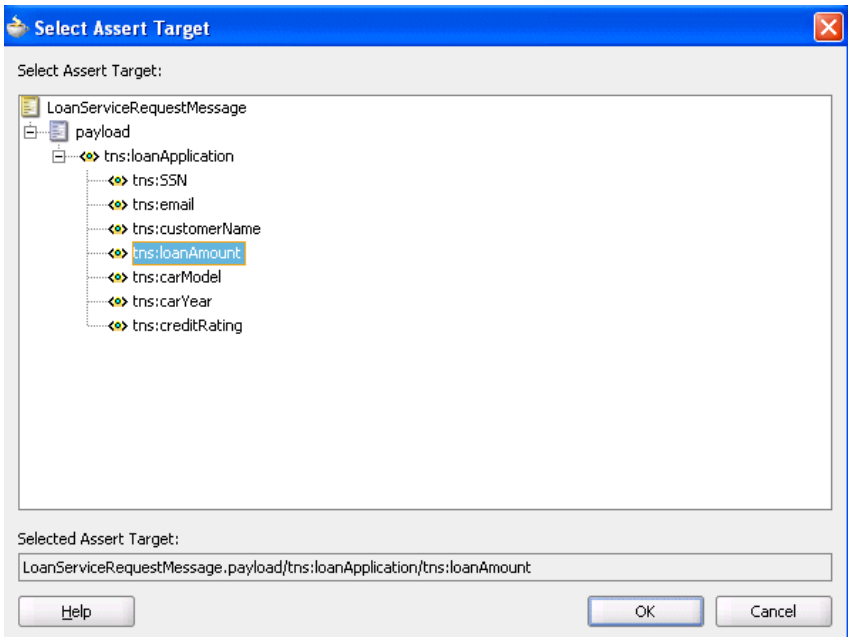
This test compares the value to an expected value.

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 50-20 provides details.

Figure 50-20 Selection of a Leaf Element



The Create Assert dialog refreshes based on your selection of an entire XML document.

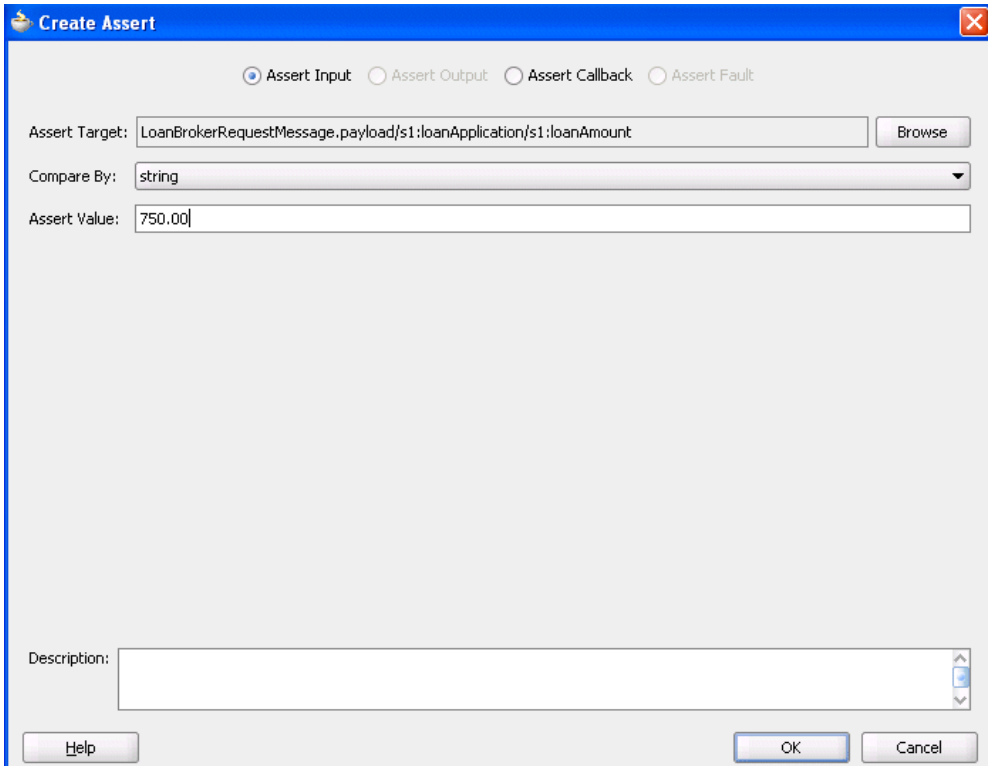
3. Enter details in the remaining fields, as shown in Table 50-13:

Table 50-13 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 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 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 50-21 shows this dialog with **Assert Input** selected:

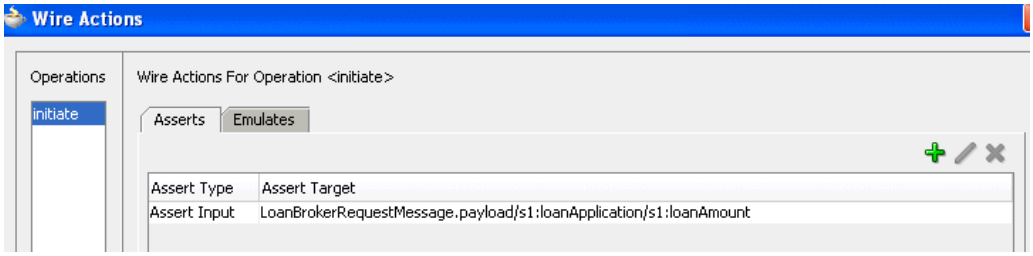
Figure 50-21 Create Assert Dialog



4. Click **OK**.

The Wire Actions dialog shown in Figure 50-22 displays your selection.

Figure 50-22 Wire Actions Dialog with Asserts Tab Selected



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.

Testing BPEL Process Service Components

After creating the basic contents of test suites and test cases with the Create Composite Test Wizard, you can automate the testing of an individual BPEL process service component included in a new or existing SOA composite application test suite. These test cases enable you to simulate the interaction between a BPEL process and its web

service partners before deployment in a production environment. This helps to ensure that a BPEL process interacts with web service partners as expected by the time it is ready for deployment to a production environment.

The following provides an example of a SOA composite application test suite that includes a component test for the `LoanBroker` BPEL process service component.

```
<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@example.com</email>
            <customerName>Joe Smith</customerName>
            <loanAmount>20000</loanAmount>
            <carModel>Camry</carModel>
            <carYear>2007</carYear>
            <creditRating>800</creditRating>
          </loanApplication>
        </content>
      </part>
    </message>
  </initiate>
  <componentTest componentName="LoanBroker" filePath="assert.xml"/>
</compositeTest>
```

The `assert.xml` test shown in the preceding example specifies assertions for variables and faults.

Note:

You cannot automate the testing of business rule, human task, Oracle Mediator, or spring service components.

Overview of Assertions on BPEL Process Activities

You can create variable and fault assertions on BPEL process activities. The following example instructs the BPEL process to ensure that the contents of `textVar` and `crOutput` match the contents specified:

```
 <bpelTest xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xmlns="http://xmlns.oracle.com/sca/2006/test"
              componentName="LoanBroker">
 <activityActions activityName="elementAssign">
    <assert comparisonMethod="number">
      <description>Some other assertion.</description>
      <expected>
        <location key="textVar"
                  xmlns:loan="http://www.autoloan.com/ns/autoloan"/>
        <simple>111222333</simple>
      </expected>
    </assert>
  </activityActions>
 <activityActions activityName="invokeCR">
    <assert comparisonMethod="number">
```

```
      <description>Make sure we got the output.</description>
      <expected>
        <location key="crOutput" partName="payload" xpath="/tns:rating"
 xmlns:tns="http://services.otn.com"/>
        <simple>560</simple>
      </expected>
    </assert>
  </activityActions>
</bpelTest>
```

For more information about creating assertions on BPEL process activities, see How to Create Assertions.

Overview of a Fast Forward Action on a Wait Activity

A wait activity allows a process to wait for a given time period or until a time limit has been reached. When testing a BPEL process service component, you may want to bypass the wait activity to continue with testing. A fast forward action enables you to specify the amount of time for which to bypass a wait activity and move forward in the test scenario. The following example instructs the BPEL process to bypass the wait activity for 1 second.

```
<bpelTest xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:schemaLocation="http://xmlns.oracle.com/sca/2006/test
 TestFwk.xsd"
               xmlns="http://xmlns.oracle.com/sca/2006/test"
               componentName="LoanBroker">
    <activityActions activityName="wait1">
      <fastForward duration="PT1S"/>
    </activityActions>
</bpelTest>
```

For more information about creating fast forward actions on wait activities, see How to Bypass a Wait Activity.

Overview of Assert Activity Execution

You can specify and validate the number of times an activity is executed in a BPEL process. The following example instructs the BPEL process to execute the `invoke`, `elementAssign`, `invokeCR`, and `replyOutput` activities one time each.

```
<bpelTest xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xmlns="http://xmlns.oracle.com/sca/2006/test"
               componentName="LoanBroker">
    <assertActivityExecuted activityName="invoke" executionCount="1"/>
    <assertActivityExecuted activityName="elementAssign" executionCount="1"/>
    <assertActivityExecuted activityName="invokeCR" executionCount="1"/>
    <assertActivityExecuted activityName="replyOutput" executionCount="1"/>
</bpelTest>
```

For more information about creating assert activity executions, see How to Specify the Number of Times to Execute an Activity.

How to Create BPEL Process Service Component Tests

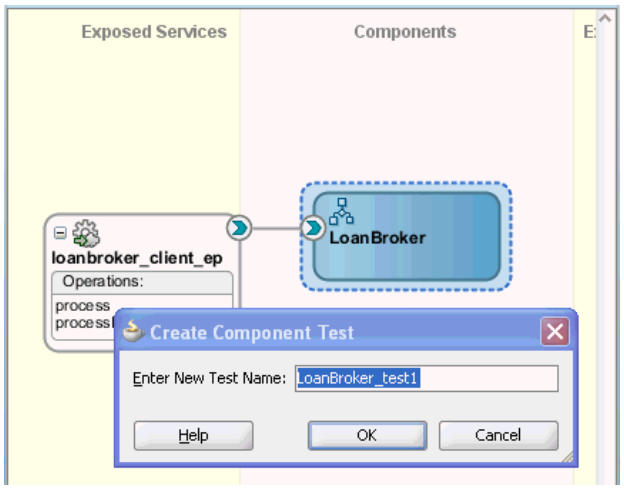
To create BPEL process service component tests:

1. Double-click a BPEL process in a test suite (for this example, **LoanBroker**).

If you have not yet created a test suite, see Creating Test Suites and Test Cases with the Create Composite Test Wizard. The BPEL process service component test that you create is included in the overall test suite for the SOA composite application.

The Create Component Test dialog is displayed, as shown in Figure 50-23.

Figure 50-23 Create Component Test Dialog



2. Accept the default name or enter a different name, as shown in Figure 50-23.
3. Click **OK**.

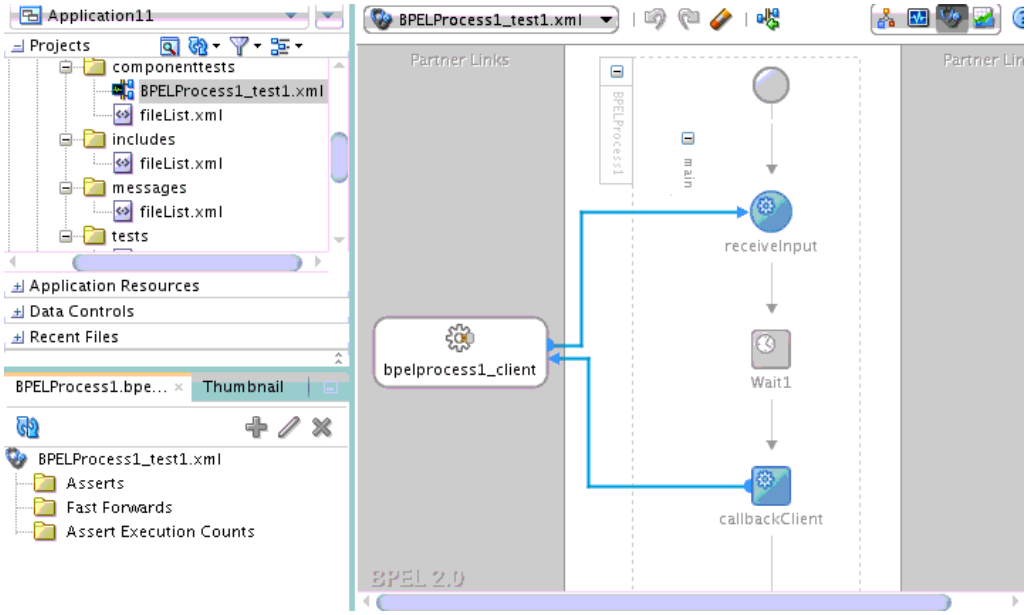
   The BPEL process in test mode is displayed, as shown in Figure 50-24.

   In the lower left section, the Structure window displays the **Asserts**, **Fast Forwards**, and **Assert Execution Counts** folders. You can right-click these folders to create assertions, fast forwards (to bypass executions of wait activities), and assertion execution counts, respectively.

   Above the designer, the following buttons are displayed:

   - **BPEL**: Click to access the BPEL process service component in design mode of Oracle BPEL Designer (that is, in nontest mode). This button is currently enabled in Figure 50-24 because you are in test mode for the BPEL process.
   - **Monitor**: Click to configure BPEL process monitors in Oracle BPEL Designer. BPEL process monitors can send data to Oracle BAM for analysis and graphical display through the Oracle BAM adapter.
   - **Test**: This button is currently disabled because you are in test mode for the BPEL process service component. This button is enabled when you click the **BPEL** button to enter design mode in Oracle BPEL Designer.
   - **Analytics**: Click to create a uniform measurement mechanism across Oracle SOA Suite components such as Oracle BPMN, human workflow, and BPEL processes for collecting disparate data.

Figure 50-24 BPEL Process Service Component in Test Mode



How to Create Assertions

You can create assertions for variables and faults in BPEL process activities.

To create assertions:

1. Select the activity on which to create an assertion through one of the following methods:

   a. In the Structure window, right-click the **Asserts** folder and select **Create**, or select the **Asserts** folder and click the **Add** button.

      The Assert dialog is displayed.

   b. In the **Activity Name** field, click the **Browse** icon to select an activity.

   or

   a. Right-click a specific BPEL activity in the designer, and select **Edit Activity Test Data**.

   b. Click the **Asserts** tab.

   c. Click the **Add** icon.

      The activity you selected is displayed in the **Activity Name** field.

2. Enter details in the remaining fields, as shown in Table 50-14.

Table 50-14 Assertions on BPEL Activities

Field	Value
Assert Variable	Select to assert a variable.
Assert Fault	Select to assert a fault.

Table 50-14 (Cont.) Assertions on BPEL Activities

Field	Value
Target	Select a target to assert: • If you selected **Assert Variable**, click the **Browse** icon to select the type of variable to assert (for example, **/autoloan:loanApplication/autoloan:SSN**). • If you selected **Assert Fault**, click the **Browse** icon to select the type of fault to assert (for example, **NegativeCredit**).
Compare By	If comparing XML documents, specify the strictness of the comparison: • **XML Identical**: Use 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**: Use 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. If comparing variables, specify the type: • **String**: Select to compare string values. • **Pattern Match Using Java Regular Expressions**: Select to compare 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. • **Number**: Select to compare numeric values.
Parts	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 Instance Sample** icon enables you to automatically generate a sample file for testing. Click the **Save As** icon 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 Applications window.

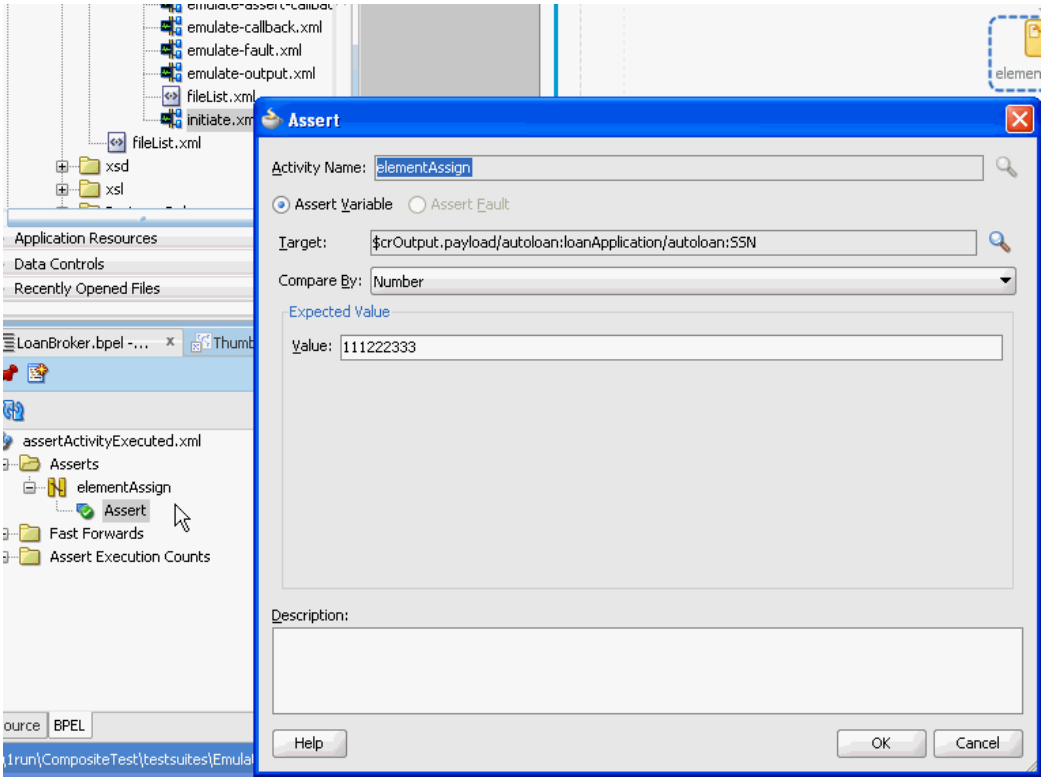
Table 50-14 (Cont.) Assertions on BPEL Activities

Field	Value
Description	Enter an optional description.

3. Click **OK**.

Expand the **Assert** folder in the Structure window to view the activities on which you have created asserts. Figure 50-25 provides details.

Figure 50-25 Assert Folder in Structure Window



How to Bypass a Wait Activity

You can specify the amount of time for which to bypass a wait activity and move forward in the test scenario. Once the time limit expires, the wait activity is processed.

To bypass a wait activity:

1. Select the wait activity to bypass through one of the following methods:

   a. In the Structure window, right-click the **Fast Forwards** folder and select **Create**, or select the **Fast Forwards** folder and click the **Add** button.

      The Fast Forward dialog is displayed.

   b. In the **Activity Name** field, click the **Browse** icon to select the wait activity.

   or

   a. Right-click a specific wait activity in the designer, and select **Edit Activity Test Data**.

b. Click the **Fast Forward** tab. This tab is only displayed if there are wait activities in the BPEL process.

c. Click the **Add** icon.

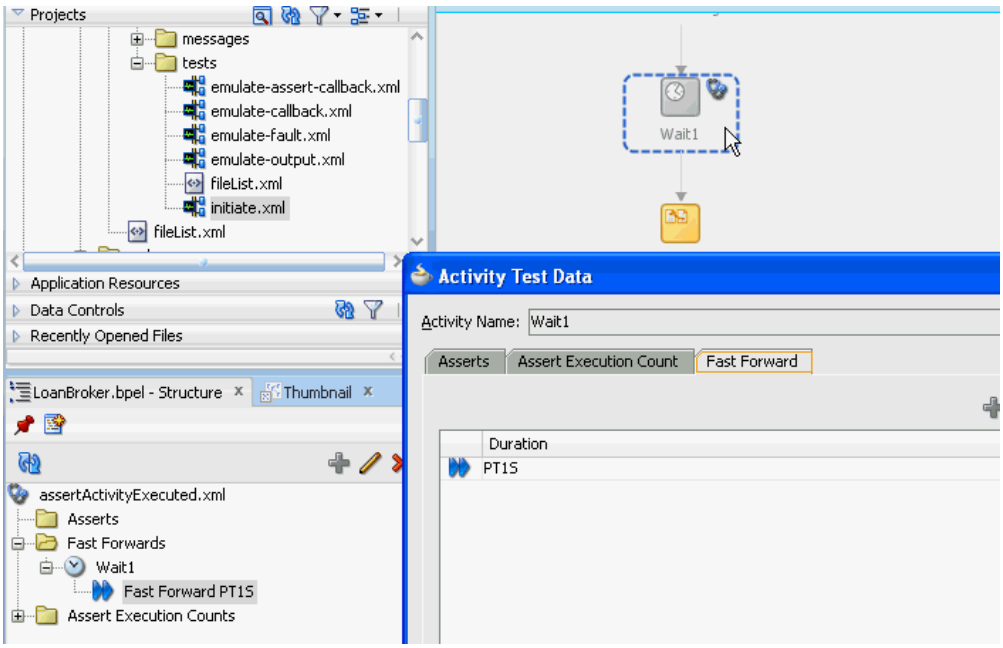
   The wait activity you selected is displayed in the **Activity Name** field.

2. In the **Duration** list, specify a time period for which to bypass the wait activity (for example, `1` second).

3. Click **OK**.

4. Expand the **Fast Forwards** folder in the Structure window to view the amount of time for which to bypass the wait activity and move forward in the test scenario. Figure 50-26 provides details.

Figure 50-26 Fast Forwards Folder in Structure Window



For more information about wait activities, see Setting an Expiration Time with a Wait Activity .

How to Specify the Number of Times to Execute an Activity

You can specify to execute an activity a specified number of times. This provides a method for verifying that an activity executes the correct number of times in a process flow (for example, ensuring that a while activity executes the correct number of times).

To specify the number of times an activity is executed:

1. Select the activity to execute through one of the following methods:

   a. In the Structure window, right-click the **Assert Execution Counts** folder and select **Create**, or select the **Assert Execution Counts** folder and click the **Add** button.

      The Assert Execution Count dialog is displayed.

b. In the **Activity Name** field, click the **Browse** icon to select the activity to execute.

or

a. Right-click a specific BPEL activity in the designer, and select **Edit Activity Test Data**.

b. Click the **Assert Execution Count** tab.

c. Click the **Add** icon.

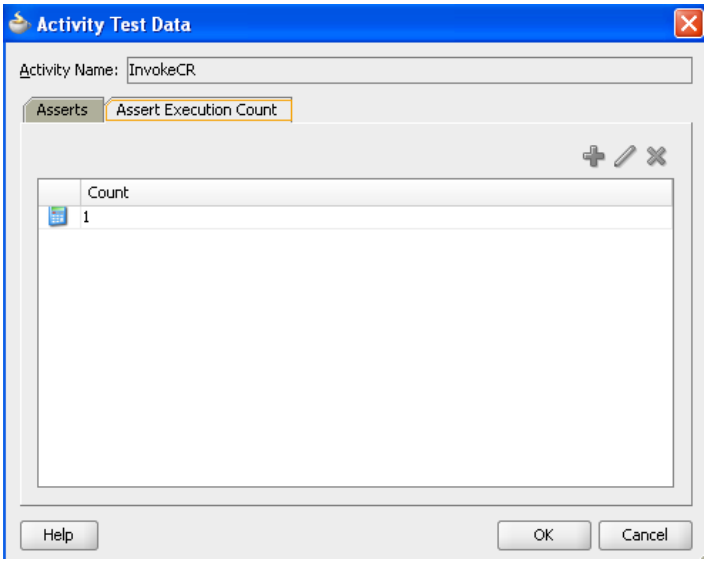
The activity you selected is displayed in the **Activity Name** field.

2. In the **Count** list, select a value.

3. Click **OK**.

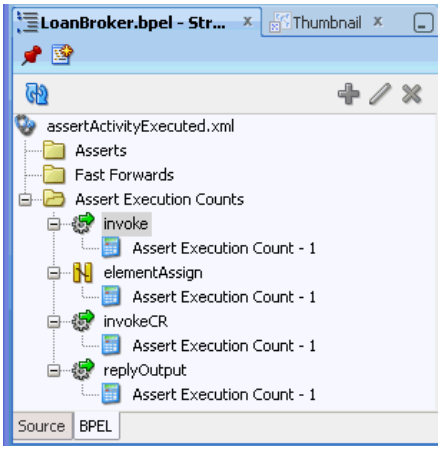
The Activity Test Data dialog looks as shown in Figure 50-27.

Figure 50-27 Activity Test Data Dialog



4. Expand the **Assert Execution Counts** folder in the Structure window to view execution counts assigned to activities. Figure 50-28 provides details.

Figure 50-28 Assert Execution Counts Folder in the Structure Window



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 JDeveloper, Oracle Enterprise Manager Fusion Middleware Control, an Oracle WebLogic Scripting Tool (WLST) script, or an `ant` command.

How to Deploy and Run a Test Suite from Oracle JDeveloper

You can run a test suite from Oracle JDeveloper. After test suites are created, you can select multiple test suites to run, an individual test suite to run, or an individual test in a test suite to run.

To deploy and run a test suite from Oracle JDeveloper:

1. Perform the appropriate task shown in Table 50-15.

Table 50-15 Test Suite Execution Options

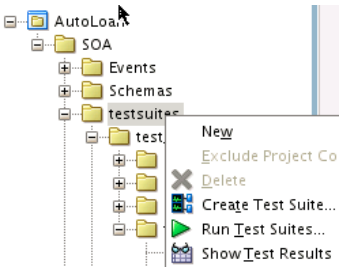
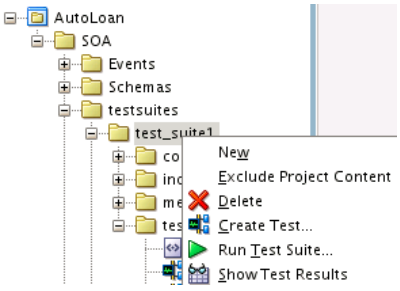
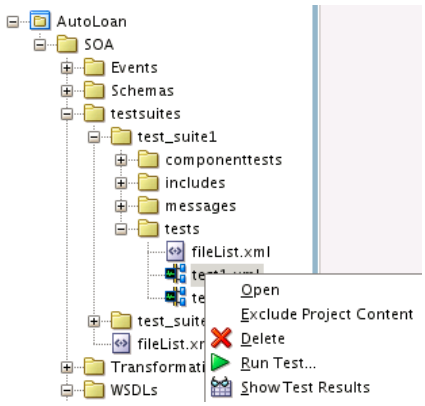
To...	In the Applications Window...
Run the test suite currently open in test mode in the SOA Composite Editor.	**a.** Click the **Run Test** icon above the SOA Composite Editor. 
Run all test suites.	**a.** Right-click the **testsuites** folder, and select **Run Test Suites**. 
Run an individual test suite.	**a.** Right-click the test suite name, and select **Run Test Suite**. 

Table 50-15 (Cont.) Test Suite Execution Options

To...	In the Applications Window...
Run an individual test in a test suite.	**a.** Right-click the individual test in the **tests** folder, and select **Run Test**. 

If you have not configured the test server to use, the Specify Test Server dialog is displayed.

2. Enter the test server host name and optionally select the **Do not ask again, save it in Tools-> Preferences-> SOA** check box. This prevents this dialog from being displayed again until you go to **Tools** > **Preferences** > **SOA** and change the configuration.

3. Click **OK**.

   The Test Run dialog is displayed.

4. Perform the following steps:

   a. Specify the test run name.

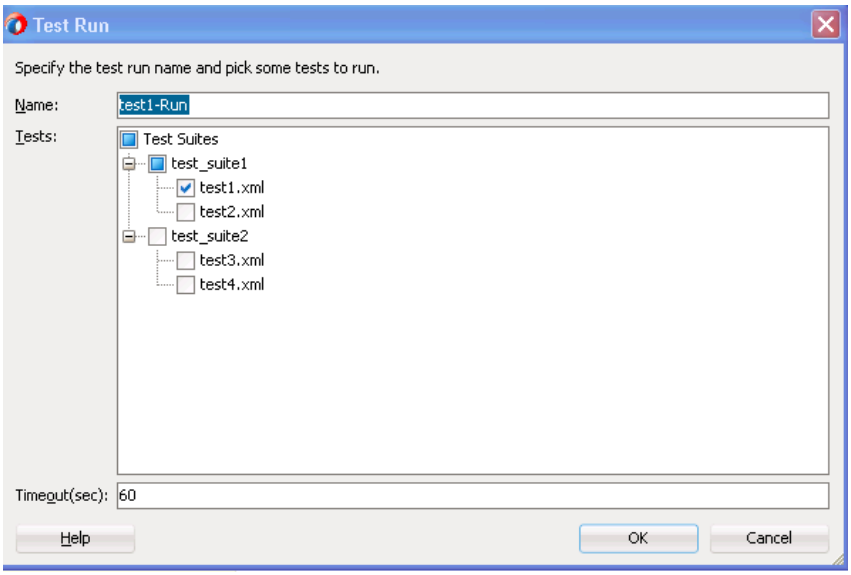
   b. Select or deselect tests to run.

   c. Specify the timeout value in seconds for running tests on the test server.

   d. Click **OK**.

   Figure 50-29 provides details.

Figure 50-29 Test Run Dialog



A check is made to see if the SOA composite application (including the tests) has ever been deployed on the test server. You must first deploy the composite before you can run tests on the test server.

5. Perform the steps shown in Table 50-16 based on the deployment status of the SOA composite application.

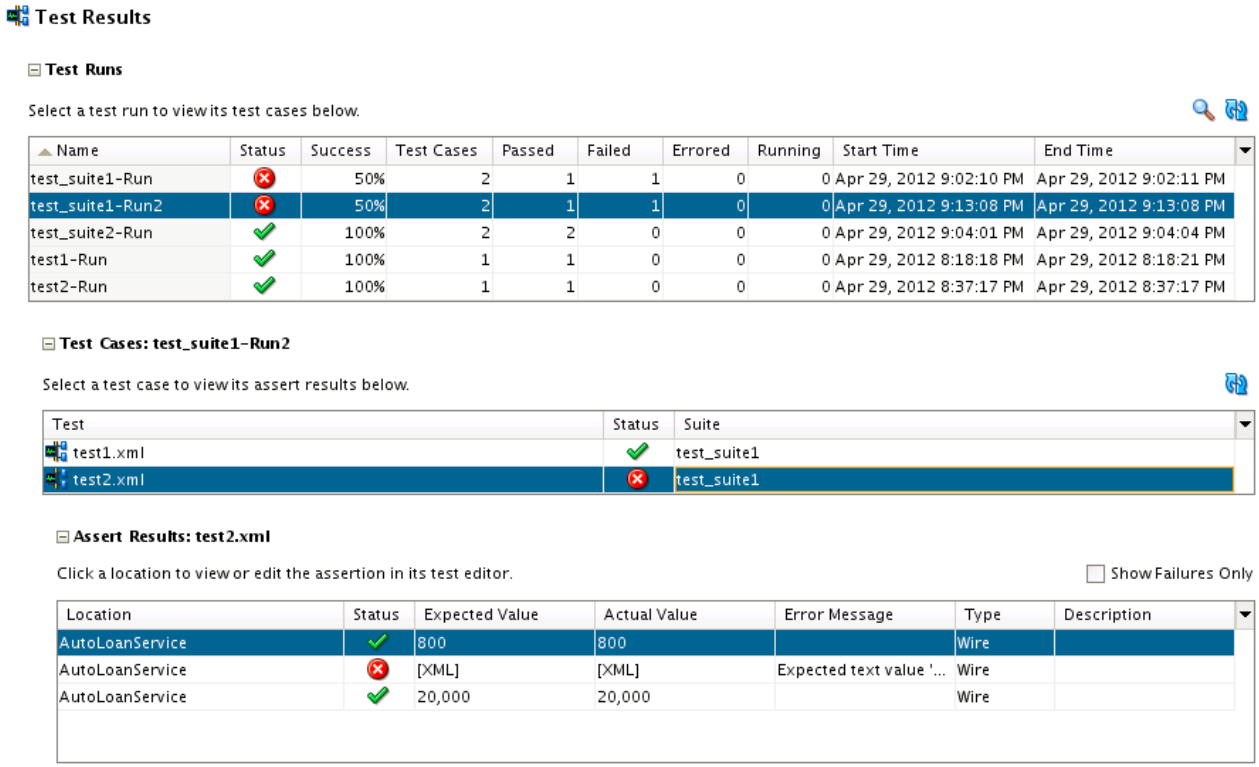
Table 50-16 Check to Determine if the SOA Composite Application is Deployed

If the SOA Composite Application ...	Then ...
Is deployed.	Go to Step 6.
• Has never been deployed on the test server. • Has been deployed on the test server, but the composite (including the tests) has been changed since the last deployment.	The Confirm to Deploy Composite dialog is displayed. **a.** Click **OK** to deploy the SOA composite application. The Deployment Action page of the Deploy *Project_Name* wizard is displayed. **b.** Select **Deploy to Application Server**. **c.** Follow the pages of the wizard to deploy the SOA composite application to an application server. For information about deploying SOA composite applications, see Deploying the Profile. **d.** When deployment is complete, go to Step 6.

After deployment has completed, the tests run on the test server.

6. View the test results. Figure 50-30 provides details. The Test Results dialog is per test server and composite DN. The test server URL (the SOA server host name and port number) and composite DN are displayed in the top right corner to indicate the context. You can run tests as many times as you want, and can select different test combinations to run on the same test server or different test servers.

Figure 50-30 Test Results Dialog

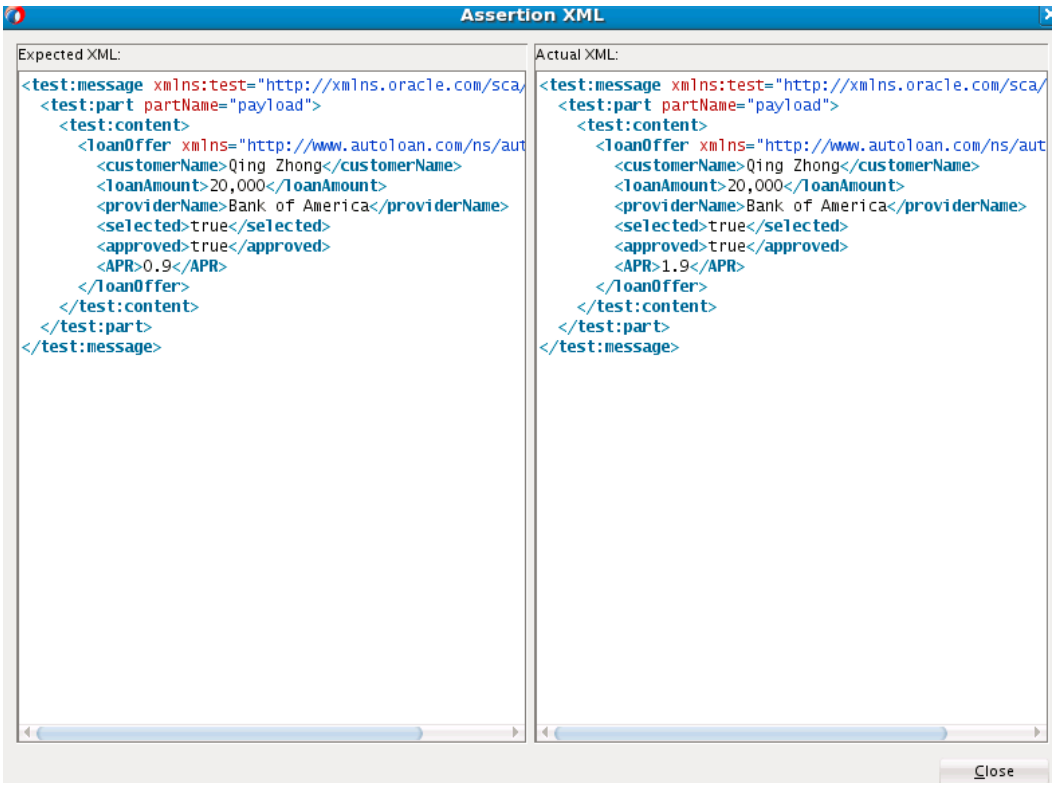


Test results are displayed in three collapsible tables, from master to details. Table 50-17 provides details.

Table 50-17 Test Results Tables

Test Runs	Test Cases	Assert Results
Shows the current test run and its status summary if you just submitted a test run. If you just queried the test server for test runs, the table shows all test runs matching your query criteria. • Name of the test run that you entered in the Test Run dialog. • Status of the test run: either passed or failed. The status is passed if all test cases in the test run passed. Otherwise, the status is failed, which means at least one test case failed. • Success percentage of the test run. • Total number of test cases. • Number of passed, failed, in error, and running test cases. • Start and end times for a test run.	Shows all test cases and the statuses of the selected test run from the **Test Runs** table. Click the **Refresh** button to refresh the test case statuses. • Test file name of the test case. Click to access its test editor. • Status of the test case, either passed or failed. The status is passed if all assertions in the test case passed. Otherwise, the status is failed, which means at least one assertion failed. • Test suite of the test case.	Shows all assertion results of the selected test case from the **Test Cases** table. • Assertion location. This is the wire source (service or reference) for a wire assert and the component (BPEL process) activity name for a component assert. This is a hyperlink to the location of the assert in its test editor. Figure 50-31 provides details. • Assertion status: Passed or failed. The status is passed if the actual value matches the expected value. • Expected and actual values of the assert. This is a simple value if it is a simple value assert and a hyperlink to a popup to show the XML value if it is an XML value assert. • Error message if the status is failed. • Assertion type: either wire or component. Wire means to assert on a composite wire. Component means to assert within a component (BPEL process). • Assertion description that you entered for the assertion when created.

Figure 50-31 Assertion XML Results



7. Perform the following additional tasks in the **Test Runs** table in Figure 50-30:

   a. Click the **Search** icon above the **Test Runs** table to query test runs from the test server by specifying search criteria.

   b. Click the **Refresh** icon above the **Test Runs** table to refresh the status of test runs.

8. Perform the following additional tasks in the **Test Cases** table in Figure 50-30:

   a. Click the **Refresh** icon above the **Test Cases** table to refresh the test case statuses.

9. Perform the following additional tasks in the **Asserts Results** table in Figure 50-30:

   a. Select the **Show Failures Only** check box above the **Asserts Results** table to show failed asserts only.

How to Deploy and Run a Test Suite from Oracle Enterprise Manager Fusion Middleware Control

For information about deploying a SOA composite application and running a test suite from Oracle Enterprise Manager Fusion Middleware Control, see *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

How to Deploy and Run a Test Suite with a WLST Command

For information about using the `sca_test` WLST command to execute a test suite, see Section "sca_test" of *WLST Command Reference for SOA Suite*.

How to Deploy and Run a Test Suite with an ant Script

For information about using the `ant-sca-test.xml` ant script to execute a test suite, see How to Use ant to Automate the Testing of a SOA Composite Application.

Part IX

Advanced Topics

This part describes advanced topics.

This part contains the following chapters:

51

Managing Large Documents and Large Numbers of Instances

This chapter describes the best practices for managing large documents and metadata and managing environments with large numbers of instances in Oracle SOA Suite. It also describes use cases for handling large documents, limitations on concurrent processing of large documents, and tuning recommendations.

This chapter includes the following sections:

- Best Practices for Handling Large Documents
- Best Practices for Handling Large Metadata
- Best Practices for Handling Large Numbers of Instances

For more information about Oracle SOA Suite tuning and performance, see *Tuning Performance*.

For information about troubleshooting Oracle SOA Suite issues, see Chapter "Troubleshooting Oracle SOA Suite and Oracle BPM Suite" of *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

For information about using Oracle Data Integrator to perform fast bulk data movement and handle complex data transformations, visit the following URL:

`http://www.oracle.com/technetwork/middleware/data-integrator`

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.

Use Cases for Handling Large Documents

This section describes use cases for handling large documents.

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.

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 51-1 provides details.

Table 51-1 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).
Oracle BPEL Process Manager/Oracle Mediator	Can be decoded in a BPEL process using Java `exec`.

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 51-2 provides details.

Table 51-2 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).
Oracle BPEL Process Manager/Oracle Mediator	No additional work is required.

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 51-3 provides details.

Table 51-3 Capabilities

Capability	Description
Security	Not supported.
Filter/Transformation/ Assign	Pass through.
Fanout	Supported.
Binding	Adapter encodes it to Base64 format.

Table 51-3 (Cont.) Capabilities

Capability	Description
Oracle BPEL Process Manager/Oracle Mediator	Supported. Opaque content cannot be manipulated in an assign or a transform activity.

Opaque Passed by Oracle B2B

In this use case, the binary attachments (for example, an image) are Base64-encoded as text encoded. Table 51-4 provides details.

Table 51-4 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.

End-to-End Streaming with Attachments

This section describes use cases for end-to-end streaming of attachments.

Note:

Direct Internet Message Encapsulation (DIME) attachments are not supported.

SOAP with Attachments

In this use case, the binary attachments (for instance, an image) are passed end-to-end as a stream. Table 51-5 provides details.

Table 51-5 Capabilities

Capability	Description
Security	Not supported.
Filter/Transformation/ Assign	Pass through. You must use an XPath extension function in Oracle BPEL Process Manager.
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. Oracle BPEL Process Manager persists it.
Tuning	Manage the database tablespace when using with Oracle BPEL Process Manager.

Table 51-5 (Cont.) Capabilities

Capability	Description

WSDL code for defining SOAP with attachments	`<mime:part>`
	`  <mime:content part="bin" type=“image/jpeg"/>`
	`</mime:part>`

Note:

- You cannot stream attachments as part of a web service callback response.
- 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. Embedded attachments refer to inlined/encoded attachments.

On the wire, messages with streamed attachments are identical to any other SOAP message with attachments.

The following example 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`).

```
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. When using Oracle BPEL Process Manager for attachments over a period, 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 about purge scripts, see *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

For information about extending tablespaces, see Section "Extending Tablespaces to Avoid Problems at Runtime" of *Administering Oracle SOA Suite and Oracle Business Process Management 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. The following example provides details.

```
<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 Section "Policy Attachments and Local Optimization in Composite-to-Composite Invocations" and Section "Configuring Local Optimization" in the *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

Note:

Oracle Web Services Manager (OWSM) does not inspect or enforce policies on streamed attachments. For more information about OWSM, see *Administering Web Services*.

Streaming Attachments from the SOA Web Service Binding Layer

You can receive the error shown in the following example when steaming attachments from the SOA web service (WS) binding layer.

```
java.lang.OutOfMemoryError: Java heap space
        at java.util.Arrays.copyOf(Arrays.java:2271)
        at java.io.ByteArrayOutputStream.grow(ByteArrayOutputStream.java:113)
        at
java.io.ByteArrayOutputStream.ensureCapacity(ByteArrayOutputStream.java:93)
        at
java.io.ByteArrayOutputStream.write(ByteArrayOutputStream.java:140)
        at
```

To resolve this error, add the following properties in the `composite.xml` file for service and reference binding components.

- `streamIncomingAttachments="true"`
- `streamOutgoingAttachments="true"`

See the preceding section for information about setting these properties.

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. The following example shows how to use this function:

```
<copy>
   <from expression="ora:getAttachmentContent('input','bin')"/>
   <to variable="initiateTaskInput" part="payload"
    query="/taskservice:initiateTask/task:task/task:attachment/task:content"/>
</copy>
```

The preceding example 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 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.

With the default configuration, Oracle Mediator can pass an attachment stream to only one target receiver, which can be another component or a web service/adapter. The second target cannot receive the attachment. When you define the `persistStreamAttachment` property for the Oracle Mediator component, Oracle Mediator can pass an attachment stream to multiple target receivers.

Oracle Mediator requires the `persistStreamAttachment` property for streaming attachments where the source message that contains the attachment is shared by

multiple target receivers. Set this property to `true` in `composite.xml` to enable the streaming of attachments to multiple targets. The following example provides details.

```
component name="Mediator1">
   <implementation.mediator src="Mediator1.mplan"/>
   <property name="persistStreamAttachment">true</property>
</component>
```

Overriding Pass Through Settings for Attachments in Oracle Mediator

Oracle Mediator automatically propagates attachments to target receivers for Oracle Mediator components that are pass through (that is, they do not contain a transformation or assign rule), and it does not propagate attachments for Oracle Mediator components that are not pass through. The `passThroughAttachment` property lets you override the pass through settings just for attachments. Setting this property to `true` copies all attachments to the target receiver implicitly.

Use this property to propagate attachments when the Oracle Mediator component is not a pass through, or use it to block attachments when the Oracle Mediator component is pass through. To implement the pass through attachment override, add the property to the project's `composite.xml` file in the `component` element for the Oracle Mediator component. Set the property to `true` to override an Oracle Mediator component that is not pass through. Set it to `false` to override a pass through component. The following example provides details:

```
<component name="Mediator">
    <implementation.mediator src="Mediator.mplan"/>
    <property name="passThroughAttachment">true</property>
</component>
```

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 the following example on all the called BPEL components (callees) in the call chain:

```
<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"
```

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 51-6 provides details.

Table 51-6 Capabilities

Capability	Description
Security	N/A.

Table 51-6 (Cont.) Capabilities

Capability	Description
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 *Understanding Technology Adapters*.

Writing Attachments Using an Outbound File Adapter

The following example shows a sample schema that can be used by the file adapter to write 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 or Oracle Mediator service component. 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 doStreamingTranslate.

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 51-7 provides details.

Table 51-7 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.

Sending and Receiving MTOM-Optimized Messages to SOA Composite Applications

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) optimized messages. 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-optimized messages.

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 a 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 Map Editor 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 SOAP with Attachments.

Scenarios for Storing SwA and MTOM-Optimized Attachments to the Database

When a SOA composite application with a BPEL process receives an MTOM-optimized SOAP message, the attachment contents of each of the MTOM-optimized elements (the ones with an `<xop href="">`) are stored in the dehydration store. Similarly, when receiving a SOAP message with attachments (SwA) message with one or more attachments, each attachment is stored in the dehydration store. These attachments can then be passed around by reference using an `href` attribute that identifies them in the database. In fact, all of the text content of these attachment elements is removed and replaced by this `href` attribute. For MTOM-optimized messages, the same value of the incoming `href` attribute from the `<xop>` element is reused. Similarly, for SwA, the `href` attributes of the attachment elements are reused.

The attachments are stored in the dehydration store when the message is delivered to the BPEL process service engine. (when the incoming message is saved into the `DLV_MESSAGE` table). Therefore, it is applicable only for one-way and asynchronous BPEL processes with `bpel.config.oneWayDeliveryPolicy` set to `async.persist` (the default value) in the `composite.xml` file.

Attachments are not persisted in the following use cases:

- If the SOAP message was received by a synchronous BPEL process or a one-way/asynchronous BPEL process with `bpel.config.oneWayDeliveryPolicy` set to `sync` or `async.cache`.
- Contents of all elements within the SOAP request with inline binary content are not persisted, but passed as-is. (That is, they do not have a child element `<xop:Include>`, but do have a base64 encoded string as a child.) An MTOM-optimized message can be a mix of one or more elements that have inline base64 data, and one or more elements that are XOP-packaged, at any level.

Note:

Even if the service binding component is MTOM-enabled, it does not automatically indicate that the service receives MTOM-optimized messages. The calling service/application must send MTOM-optimized messages over the wire to ensure the message is received. MTOM-enabled bindings can also receive ordinary non-MTOM messages. Therefore, when it receives one, the SOAP requests arriving into the service can have nonoptimized inline binary data elements that are not be persisted into the database.

Even though the content of the MTOM-optimized elements or SwA attachments have their value replaced by an `href` attribute at runtime, their design-time WSDLs still remain unaltered. You do not see these changes in Oracle JDeveloper. Their element type definitions do not change from hexBinary, base64Binary, and so on to that of an empty content with an `href` attribute.

However, this is transparent to you. For instance, when you use an assign activity to copy across their content, the `href` values are copied over at runtime. Similarly, when invoking an outbound reference such as a web service or an adapter, Oracle SOA Suite automatically resolves the `href` attribute to the actual data and executes the invocation.

Processing Large XML with Repeating Constructs

This section describes use cases for processing large XML with repeating constructs.

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 51-8 provides details.

Table 51-8 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 *Understanding Technology Adapters*.

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 51-9 provides details.

Table 51-9 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 *Understanding Technology Adapters*.

Processing Large XML Documents with Complex Structures

This section describes use cases for processing very large XML documents with complex structures.

Streaming with the File/FTP Adapters

In this use case, very large XML files are streamed through Oracle SOA Suite. Table 51-10 provides details.

Table 51-10 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 *Understanding Technology Adapters*.

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 51-11 provides details.

Table 51-11 Capabilities

Capability	Description
Security	N/A.
Filter/Transformation/ Assign	Supported, but must optimize to avoid issues.

Table 51-11 (Cont.) Capabilities

Capability	Description
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.

Limitations on Concurrent Processing of Large Documents

This section describes the limitations on concurrent processing of large documents.

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.

JVM Memory Sizing Recommendations for SOA Composite Applications

Sending messages with payloads that are 100 MB or larger in size can exceed JVM heap size limits if not correctly tuned.

For example, when sending large payloads in the event delivery network (EDN) with Oracle advanced queueing (AQ) JMS, ensure that you set the maximum memory value by first testing with a typical message payload size and a maximum potential message size. Using a lesser memory value can result in an `ORACLE.JMS.AQJMSEXCEPTION` error. For example, to send a payload of 100 MB, it is recommended that you change the JTA time out and maximum memory to 5 GB.

For more information about tuning the JVM heap size, see General Tuning Recommendations and Section "Java HotSpot VM Heap Size Options" of *Tuning Performance of Oracle WebLogic Server*.

General Tuning Recommendations

This section provides general tuning recommendations.

For more information about Oracle SOA Suite tuning and performance, see *Tuning Performance*.

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 *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.
- In Oracle Enterprise Manager Fusion Middleware Control, set the audit level to **Off** or **Production** at the SOA composite application level. See Setting Audit Levels from 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 a `ResourceDisabledException` error 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 XPath Extension Functions.

- Extend data files for handling large attachments. For more information, see the .

- Increase the HTTP POST timeout for `SocketException: Broken pipe` errors in Oracle WebLogic Server Administration Console. See Increasing the HTTP POST Timeout

- 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.

  For more information, see Increasing the Timeout Value.

Increasing the HTTP POST Timeout

Increase the HTTP POST timeout for `SocketException: Broken pipe` errors in Oracle WebLogic Server Administration Console.

1. From the **Domain Structure**, select **soainfra** > **servers** > *server_name* > **Protocols** > **HTTP**.
2. In the **Post Timeout** field, enter `120` (maximum).

Increasing the Timeout 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**

To increase the Stuck Thread Max Time property value:

Follow the instructions in Chapter "Using the WebLogic 8.1 Thread Pool Model" of *Tuning Performance of Oracle WebLogic Server*.

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**, 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 *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

Using the Assign Activity in Oracle BPEL Process Manager and 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.

Using XSLT Transformations on Large Payloads (For 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 was 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 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>
   ```

Using XSLT Transformations on Large Payloads (For Oracle Mediator)

Until 11*g* Release 1 11.1.1.3, for XSLT operations in Oracle Mediator, 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, the `streamResultToTempFile` property was 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 payloads using XSLT. The default value is `no`.

Note:

This property is recommended only for processing large payloads. Enabling this property could reduce performance for normal payloads.

To configure large XML documents to be processed using XSLT:

1. Create an Oracle SOA Suite project with an Oracle Mediator component.
2. Open the `composite.xml` file for the project in **Source** view.
3. In the `composite.xml` file, scroll to the component element that defines the Oracle Mediator component to process large XML documents, and add the `streamResultToTempFile` property. Set the property to `yes` as shown below.

```
<component name="Mediator1">
   <implementation.mediator src="Mediator1.mplan"/>
   <property name="streamResultToTempFile">yes</property>
</component>
```

4. Save and close the file.

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 transformations 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` or `med:doTranslateFromNative`
- `ora:doTranslateToNative` or `med:doTranslateToNative`
- `ora:doStreamingTranslate` or `med:doStreamingTranslate`

For more information about these functions, see XPath Extension Functions and *Understanding Technology Adapters*.

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`
- Large payload size

The following sections describe the parameters you must set for processing large documents in Oracle B2B. For more information, see Section "Using Document Streams to Handle Large Payloads" of *User's Guide for Oracle B2B*.

MDSInstance Cache Size

To set the Oracle Metadata Services (MDS) Repository instance cache size, use Oracle Enterprise Manager Fusion Middleware Control. This property depends on the size of the metadata. Specify a value based on the metadata/endpoint count. The default value is `100000`. For information, see Section "Setting B2B Configuration Properties in Fusion Middleware Control" of *User's Guide for Oracle B2B*.

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:

To configure the protocol message size:

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 the following example:

```
<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.

Number of Threads

This parameter improves the message processing capability of Oracle B2B and must be set in the Oracle Enterprise Manager Fusion Middleware Control. For more

information, see Section "Configuring Oracle B2B Server Properties" of *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

- **b2b.inboundThreadCount**
- **b2b.inboundSleepTime**
- **b2b.outboundThreadCount**
- **b2b.outboundSleepTime**
- **b2b.defaultThreadCount**

Stuck Thread Max Time Parameter

The **Stuck Thread Max Time** parameter checks the number of seconds that a thread must continually work before the server considers the thread stuck. You must change the following setting in the Oracle WebLogic Server Administration Console:

To configure the Stuck Thread Max Time parameter:

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**.

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 the following example:

```
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;
```

Setting a Size Restriction on Inbound Web Service Message Size

If you want to set a size restriction on inbound web service message size, configure the binding component property `max-message-size` in the `composite.xml` file. The property value is made available to the underlying web service infrastructure, which uses the value to test against the incoming message size. If the value specified is exceeded, an exception is thrown indicating that the message size is too large and the transaction is not processed. The following example provides details:

```
<composite name="LrgMsg" revision="1.0" label="2011-09-08_22-53-53_259"
 mode="active" state="on">
  <import namespace="http://xmlns.oracle.com/LargeMsg/LrgMsg/BPELProcess1"
   location="BPELProcess1.wsdl" importType="wsdl"/>
  <service name="bpelprocess1_client_ep" ui:wsdlLocation="BPELProcess1.wsdl">
     <interface.wsdl
 interface="http://xmlns.oracle.com/LargeMsg/LrgMsg/BPELProcess1# wsdl.interface
(BPELProcess1)"/>

<binding.ws port="http://xmlns.oracle.com/LargeMsg/LrgMsg/BPELProcess1
    #wsdl.endpoint(bpelprocess1_client_ep/BPELProcess1_pt)">
```

```
  <property name="max-message-size" type="xs:integer" many="false"
    override="may">4</property>
</binding.ws>
</service>

  <component name="BPELProcess1" version="1.1">
    <implementation.bpel src="BPELProcess1.bpel"/>
  </component>

  <wire>
    <source.uri>bpelprocess1_client_ep</source.uri>
    <target.uri>BPELProcess1/bpelprocess1_client</target.uri>
  </wire>
</composite>
```

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 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.

- `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 the example that follows. 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 temporary file, leave this property set to its default value of `no`.

```
<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 about using this function, see 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 about using the function, see doXSLTransformForDoc.

Best Practices for Handling Large Metadata

This section provides recommendations for handling large metadata.

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

Keep the number of incoming request messages at a proper level to ensure system memory stability.

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.

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 Customizing the Number of Flow Activities with the flowN Activity in BPEL 1.1 and Section "Configuring BPEL Process Service Engine Properties" of *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

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.

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 process compilation fails.

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 process compilation may fail for 6000 activities.

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.

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`: Purges the entire repository.
- `import`: Imports the specified ZIP file.
- `deploy`: Deploys 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 *User's Guide for Oracle B2B*.

Best Practices for Handling Large Numbers of Instances

This section provides recommendations for handling large numbers of instance and fault metrics.

Instance and Rejected Message Deletion with the Purge Script or Oracle Enterprise Manager Fusion Middleware Control

You can delete thousands of instances and rejected messages with the PL/SQL purge script or from the Auto Purge page in Oracle Enterprise Manager Fusion Middleware Control.

For more information, see *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

52

Customizing SOA Composite Applications

This chapter describes how to customize SOA composite applications with the customization feature available with a BPEL process service component. It describes how to create a customizable application, customize the vertical version of the application, and customize the customer version of the application. It also describes how to upgrade to the next version of the application.

This chapter includes the following sections:

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.

Layer values are the values for a given customization layer. It is a one-to-many relationship from a layer to its layer values. You select a layer value from a layer to perform customizations. For example, assume you specify a customization class representing a customization layer called `Country`. You can then specify countries for its values, such as `USA`, `China`, and `India`. When you restart Oracle JDeveloper in the **Customization Developer** role to perform customizations, you must select one of the layer values (that is, a country) of the layer from the Oracle JDeveloper Customization Context window such as `USA`, which means you want to create the customization for that country.

Creating the Customizable Composite

This section provides an overview of the steps required for creating the customizable, base SOA composite application.

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** > **From Gallery** > **General** > **Java**.
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");
       }
   }
   ```

   For the customization class to have the correct customization layer, the customization layer name must be set 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.

   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();
       }
   }
   ```

To make the customization class effective, compile the customization class by building the SOA project.

4. In the Applications window, right-click the SOA project and select **Build *SOA_project_name*.jpr**.
5. Ensure that the build succeeds by reviewing the output in the Log window at the bottom of Oracle JDeveloper.

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. In the Configure SOA Settings dialog of the Create SOA Application wizard (Step 3 of 3), select both **Composite With BPEL Process** and the **Customizable** check box, and click **Finish**.
5. Design the BPEL process.

Note:

If you design a transformation, note the following customization restrictions in the XSLT Map Editor:

- The **Create in Template** option that is displayed by right-clicking a node in the target panel is disabled.
- The **Test XSL Map** option is disabled for the call templates and apply templates for imported XSL files. This option works for named templates, but not for template rules with a match attribute.

---

6. Customize the BPEL process by creating a scope activity. This action is required because by default the BPEL process is not customizable.

Note:

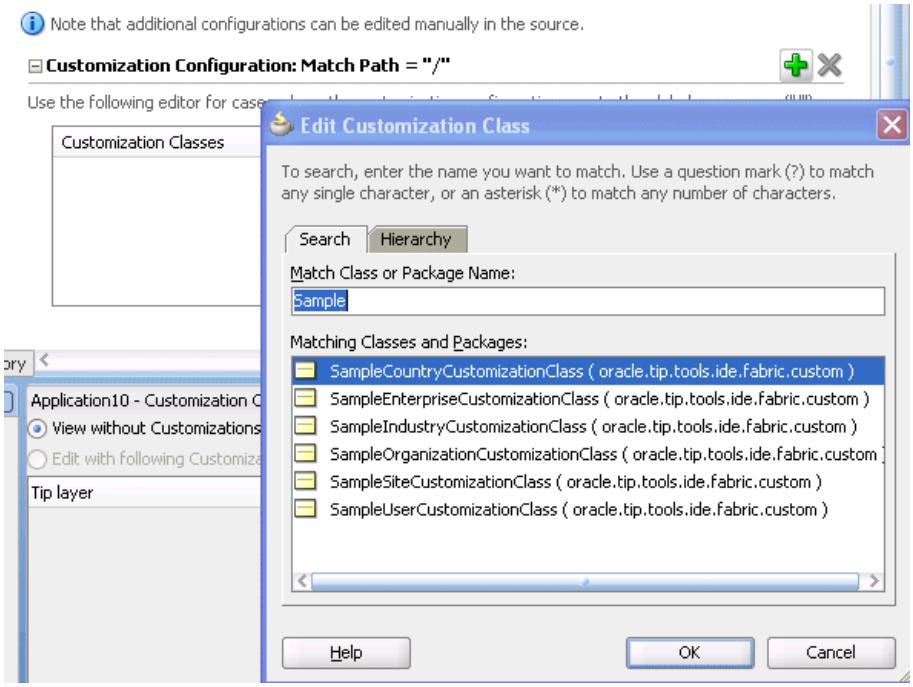
You can only customize the `composite.xml` file, `.bpel` file (for Oracle BPEL Process Manager), `.xsl` map file, and `.mplan` file (for Oracle Mediator) when logged into Oracle JDeveloper with the **Customization Developer** role.

---

7. Right-click the scope and select **Customizable**. If you expand the scope and right-click it, you do not see the **Customizable** option.
8. In the Applications window, expand **Application Resources** > **Descriptors** > **ADF META_INF**.
9. Open the **adf-config.xml** file and select the **MDS** tab.
10. Click the **Add** icon to add the required customization classes, as shown in Figure 52-1.

In real environments, the customization classes are provided by the core applications team, as described in the example scenario in Introduction to Customizing SOA Composite Applications. 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 52-1 Customization Classes



11. Right-click the SOA project and select **Deploy**.

Note:

You can receive a compilation error if your scope activity is empty. You can drag an empty activity into the scope activity to pass compilation.

---

12. On the Deployment Action page, select **Generate SAR File**. This creates a JAR file package. This JAR is also known as a SOA archive (SAR).

13. 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 *Developing Fusion Web Applications with Oracle Application Development Framework*.

How to Add an XSD or WSDL File

To add an XSD or WSDL file:

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 Applications window.

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.

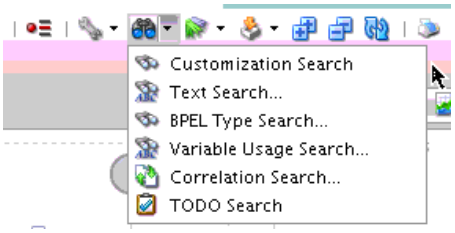
How to Search for Customized Activities in a BPEL Process

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 52-2.

Figure 52-2 Customization Search Option



The search results display in the **Search for Customizations** tab of the Log window at the bottom of the designer.

What You May Need to Know About Resolving Validation Errors in Oracle JDeveloper

In the customization role, the Oracle Metadata Services (MDS) Repository merges customizations with the base metadata. The merging can result in an invalid XML document against its schema. MDS Repository 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 customizable 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. The following example provides details:

```
<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. The following example provides details.

```
<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.

What You May Need to Know About Resolving a Sequence Conflict

This section provides an example of how to resolve a sequence conflict.

To resolve a sequence complex:

1. Customize version 1 of a SOA composite application.

   For example, while logged into Oracle JDeveloper with 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 the **Customization Developer** role.

   The following error is displayed:

   ```
   Sequence element is not expected
   ```

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.

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 the following example. 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.

```
[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.
```

Customizing the Vertical Application

This section provides an overview of the steps required for customizing the vertical SOA composite application.

Note:

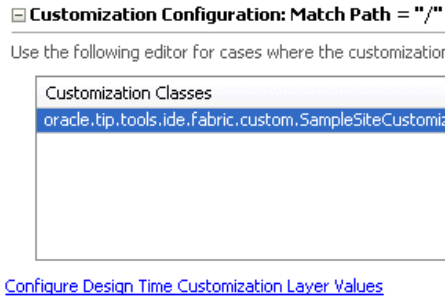
Do *not* customize the same SOA composite application for different layer values. Layer values are the customizations made to the base composite, as described in Introduction to Customizing SOA Composite Applications. Only a single layer value for customization is supported. If you must support another 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.

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 52-3.

Figure 52-3 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. The customization class provides the layer name and the **CustomizationLayerValues.xml** file provides the layer values. Both are required. 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 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.

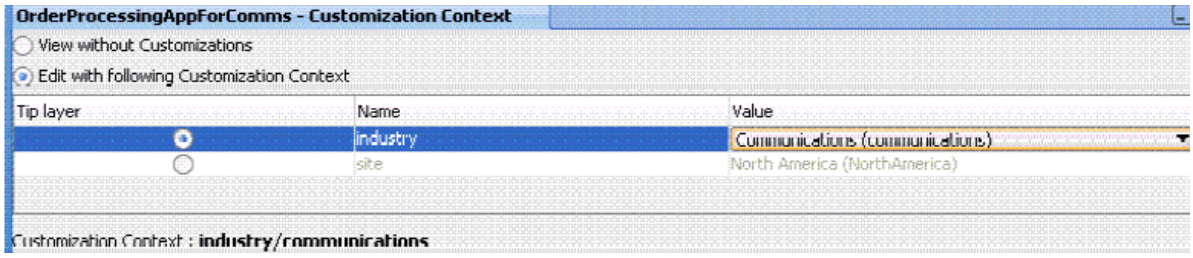
---

7. Select the **Import for Customization** check box.
8. In the Applications window, right-click the project, and select **SOA** > **Customizable**.
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 52-4 (for this example, layer **industry** and value **Communications** are selected).

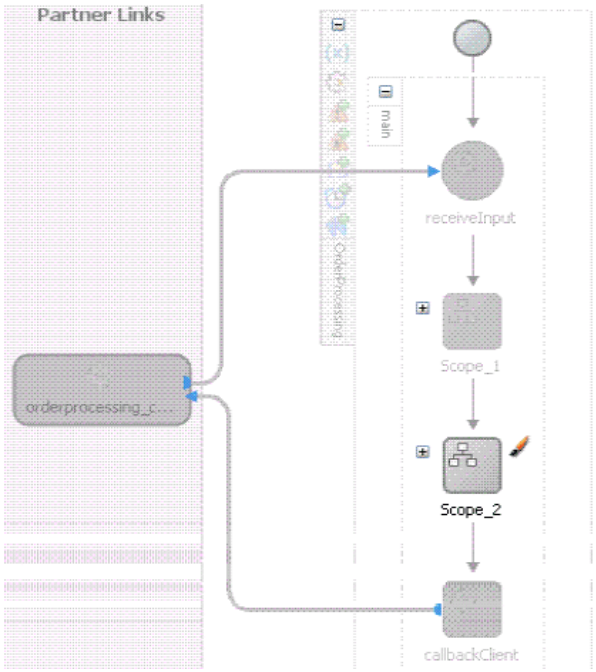
Figure 52-4 Customization Context



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 52-5, 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 52-5 One Customizable Scope



12. Right-click the SOA project in the Applications window 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 logged into Oracle JDeveloper with the **Customization Developer** role. In the Applications window, right-click the project name and select **SOA** > **Create WSDL Document** or **SOA** > **Create XML Schema**.

Customizing the Customer Version

This section provides an overview of the steps required for customizing the customer version of the SOA composite application.

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 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 Customizing the Vertical Application.

6. Select the **Import for Customization** check box.

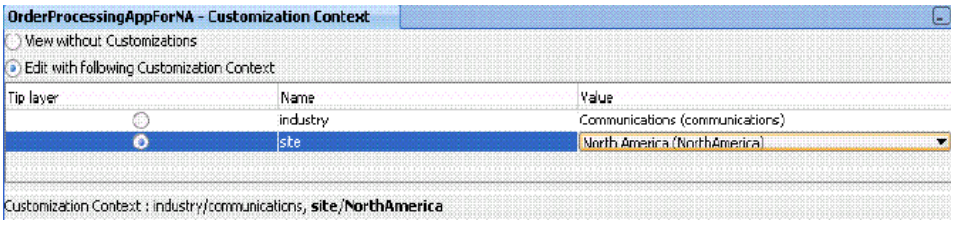
7. From the **Tools** menu, select **Switch Roles** > **Customization Developer**.

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 52-6 (for this example, the layer **site** and value **North America** are selected).

   Figure 52-6 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.

Upgrading the Composite

This section provides an overview of the steps required for upgrading the SOA composite application to the next version.

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 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.

How to Upgrade the Vertical Applications Team Composite

The vertical applications team customizes the new base composite to create a version of the JAR file.

To upgrade the vertical applications team composite:

1. Check out the application created in 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 Applications window 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 How to Upgrade the Core Application Team Composite.
6. From the **Tools** menu, select **Switch Roles** > **Customization Developer**.
7. Restart Oracle JDeveloper.
8. In the Customization Context dialog, select a layer and value to customize (for example, select the 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.

How to Upgrade the Customer Composite

The customer follows the same procedures as the vertical applications team in How to Upgrade the Vertical Applications Team Composite to apply their layers to the new base composite.

53

Defining Composite Sensors

This chapter describes how to define composite sensors that provide a method for implementing trackable fields on messages in a SOA composite application. It describes how to define sensors on binding components and on service components that have subscribed to business events. It also describes restrictions on using composite sensors and how to manage composite sensors during runtime in Oracle SOA Composer.

This chapter includes the following sections:

- Introduction to Composite Sensors
- Adding Composite Sensors
- Monitoring Composite Sensor Data During Runtime
- Creating and Managing Composite Sensors During Runtime from Oracle SOA Composer

For information about activity, fault, and variable sensors in a BPEL process, see Using Sensors and Analytics .

For examples of using composite sensors in business scenarios, see *Understanding Oracle SOA Suite*.

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 Flow Instances pages for the SOA Infrastructure, partition, and SOA composite application in Oracle Enterprise Manager Fusion Middleware Control. This action enables you to display details about a particular instance with a composite sensor.
- Publish JMS data computed from incoming and outgoing messages.
- Track composite instances initiated through business event subscriptions.

You define composite sensors on service and reference binding components or on service components that have business event subscriptions in Oracle JDeveloper. This functionality is similar to variable sensors in BPEL processes. During runtime, composite sensor data is persisted in the database.

You can also define composite sensors during runtime in Oracle SOA Composer. Oracle SOA Composer changes are picked up immediately by the runtime, whereas changes made using Oracle JDeveloper require SOA composite application redeployment.

For information about searching for composite sensors in Oracle Enterprise Manager Fusion Middleware Control, see Section "Tracking Business Flow Instances at the SOA Infrastructure or Partition Level" of *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

Restrictions on Use of Composite Sensors

Note the following restrictions on the use of composite sensors:

- Functions in XPath expressions cannot be used with properties.
- Any composite sensor that is defined by an expression always captures values as strings. This causes the sensor type to always be a string. This action makes the search possible.

  Capturing values as strings may be useful when dealing with XML types derived from a string. The following example provides details:

  ```
  <xs:element name="CardNum">
      <xs:simpleType>
          <xs:restriction base="xs:string">
              <xs:length value="16"/>
          </xs:restriction>
      </xs:simpleType>
  </xs:element>
  ```

  Even if the expression is a number, it is captured as a string. You cannot use other logical operators such as <, >, =, or any combination of these.
- Any composite sensor that is defined by a variable uses the variable type to determine the sensor type. Sensors can be one of the following types:
  - `STRING`
  - `NUMBER`
  - `DATE`
  - `DATE_TIME`
  - Complex XML
- 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.
- 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"
  ```
- Sensors cannot be configured on service components that publish business events.
- Sensors based on business event headers are not allowed (only payloads are allowed).

- PL/SQL subscriptions are not supported.

Adding Composite Sensors

You add sensors to the following components of a SOA composite application in the SOA Composite Editor:

- Service or reference binding components
- Service components such as a BPEL process or Oracle Mediator that have subscribed to business events

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 or a service component that has a subscribed business event. A service component that has a subscribed business event includes the word **Subscribed** on it.

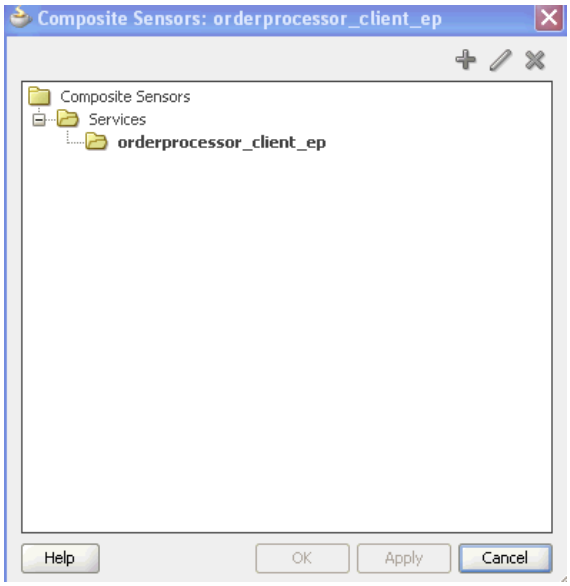
   b. Select **Configure Sensors**.

   Note:

   The service component must already have a subscribed business event for the **Configure Sensors** option to be displayed.

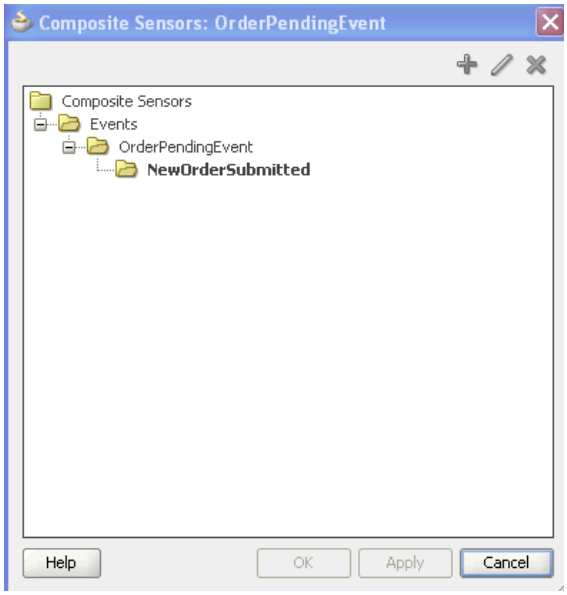
   If you selected a binding component, the Composite Sensors dialog displays the details shown in Figure 53-1. For this example, a service binding component is selected.

Figure 53-1 Composite Sensors Dialog for the Selected Binding Component



If you selected a service component, the Composite Sensors dialog displays the details shown in Figure 53-2.

Figure 53-2 Composite Sensors Dialog for the Selected Service Component

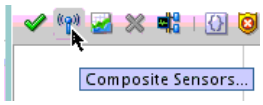


c. Select the binding component or service component in the dialog, and click the **Add** icon.

or

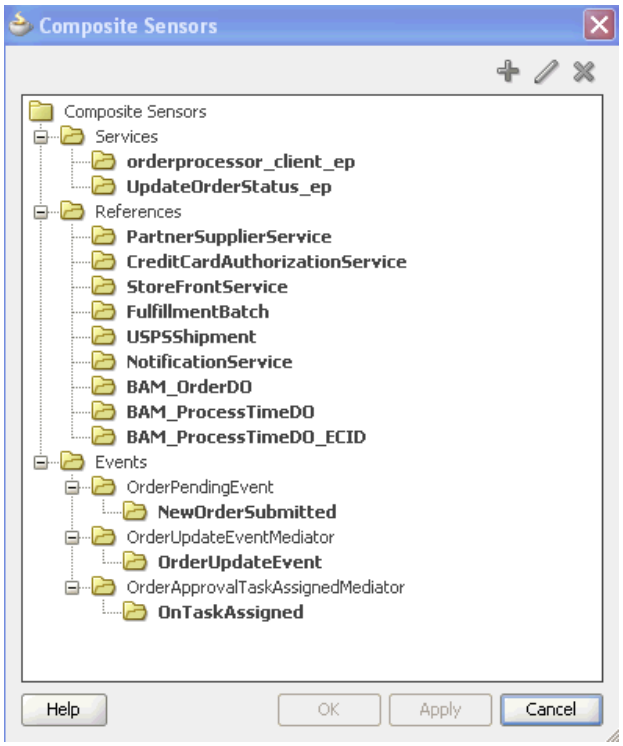
a. Click the **Composite Sensor** icon above the SOA Composite Editor, as shown in Figure 53-3.

Figure 53-3 Composite Sensor Icon



The Composite Sensors dialog for the SOA composite application appears, as shown in Figure 53-4. This option displays all the service and reference binding components and service components with subscribed business events in the SOA composite application.

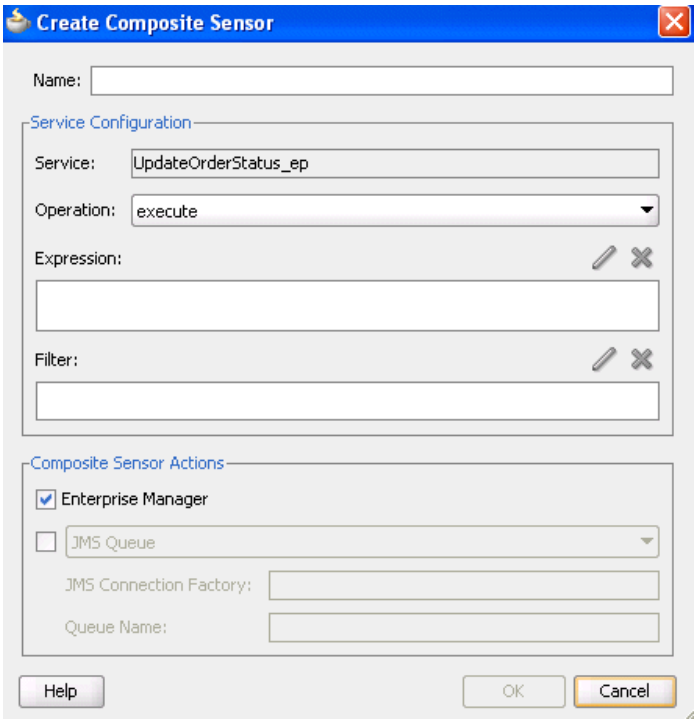
Figure 53-4 Composite Sensors Dialog



b. Select the specific service, reference, or business event to which to add a composite sensor, then click the **Add** icon.

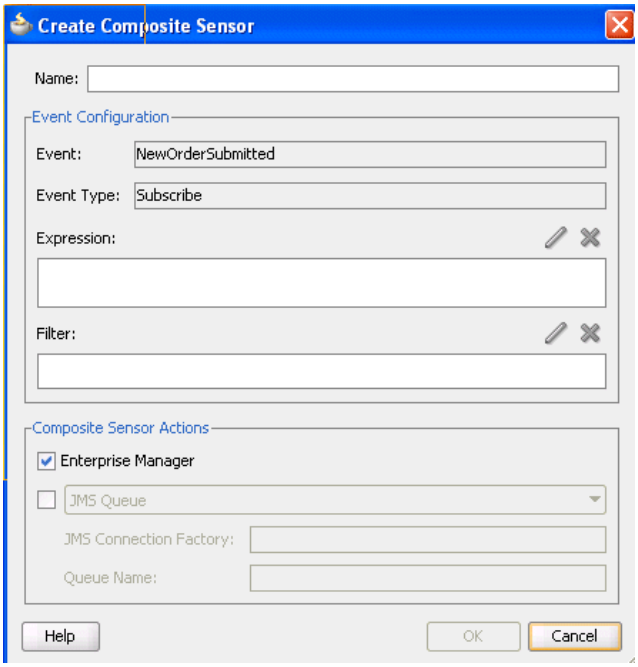
If you selected a binding component such as a service, the Create Composite Sensor dialog appears as shown in Figure 53-5.

Figure 53-5 Create Composite Sensor Dialog for a Service Binding Component



If you selected a service component that has a business event subscription, the Create Composite Sensor dialog appears as shown in Figure 53-6.

Figure 53-6 Create Composite Sensor Dialog for a Service Component



2. Enter the details shown in Table 53-1.

Table 53-1 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 is only displayed 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 is only displayed 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. This field only displays for service or reference binding components.
Event	Displays the name of the service component. This field is only displayed if you are creating a composite sensor for a service component. This field cannot be edited. Event sensors track composite instances initiated through a business event. You can create multiple sensors per business event.
Event Type	Displays the **Subscribe** business event type. This field cannot be edited. The publish business event type is not supported.
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 How to Add a Variable for instructions. • **Expression**: Select to invoke the Expression Builder dialog for creating an XPath expression. This action always captures values as strings. See 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 scope activity (in BPEL 2.0). See 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`

Table 53-1 (Cont.) Create Composite Sensor Dialog

Name	Description
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 **Flow 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 *Administering Oracle SOA Suite and Oracle Business Process Management 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 (BAM) and Oracle Complex Event Processing. Both selections use the native JMS support provided with Oracle WebLogic Server, and *not* the Oracle SOA Suite JMS adapter described in *Understanding Technology Adapters*. You can view JMS messages in the Oracle WebLogic Server Administration Console.

3. Click **OK** when complete.

For a service or reference binding component, a **composite sensor** icon displays in the upper right corner, as shown in Figure 53-7.

Figure 53-7 Sensor Icon on Binding Component

For a service component, a **composite sensor** icon also displays in the upper right corner, as shown in Figure 53-8.

Figure 53-8 Sensor Icon on Service Component

4. Place your cursor over the composite sensor icon to display details about the composite sensor.

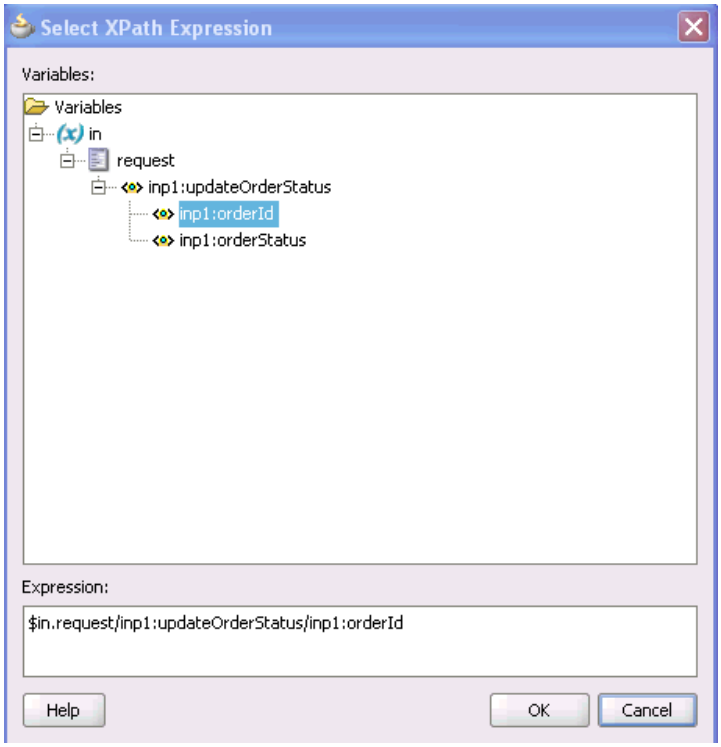
How to Add a Variable

The Select XPath Expression dialog shown in Figure 53-9 enables you to select an element for tracking.

To add a variable:

1. Expand the tree and select the element to track (for this example, an order ID).

Figure 53-9 Variables



2. Click **OK** when complete.

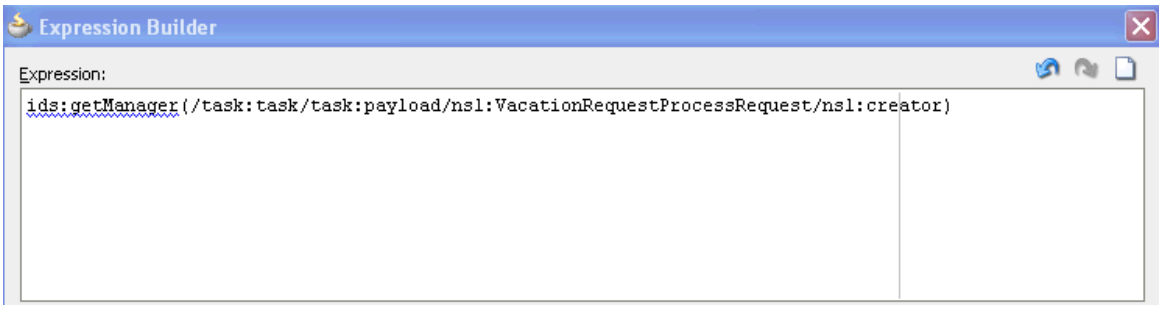
How to Add an Expression

The Expression Builder dialog shown in Figure 53-10 enables you to create an expression for tracking.

To add an expression:

1. Build an XPath expression of an element to track.

Figure 53-10 Expression



2. Click **OK** when complete.

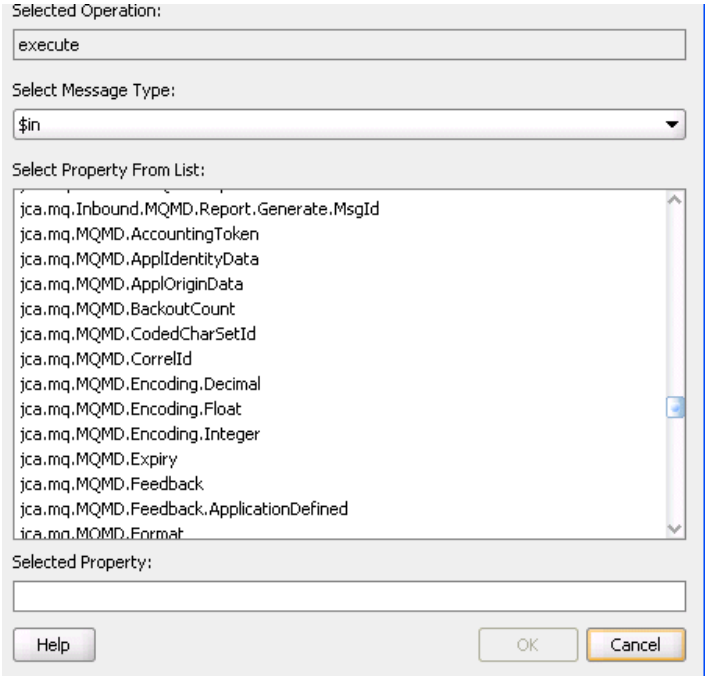
How to Add a Property

The Select Property dialog shown in Figure 53-11 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 53-11 Properties



2. Click **OK** when complete.

For more information about normalized messages, see Propagating Normalized Message Properties Through Message Headers.

What You May Need to Know About Duplicate Composite Sensor Names

Note the following details when using duplicate names for composite sensors.

- If you create composite sensors with duplicate names, the entire contents of their definitions are compared. Duplicate names are permitted where one or more additional parameters are different (for example, either different configuration types or different expressions, filters, operation names, and so on). Something must be different in the definitions for duplicate names to be permitted.
- If you have duplicate sensor definitions, only the last executed sensor value is persisted. Therefore, you can use this type of configuration for mutually exclusive paths (for example, a composite can be invoked through service 1 or service 2). Therefore, you can define the same sensor name on both the services. However, if you define the same names for service 1 and reference 1, only the sensor value from reference 1 (the last executed sensor) is stored.
- You typically use multiple sensors with the same name to point to the same logical entity extracted from different sources (for example, Oracle Enterprise Manager Fusion Middleware Control displays the final sensor value). Therefore, it can be confusing if the same sensor name is used to extract an email value and a social security value from different sources.
- Sensor actions apply to all occurrences of the same sensor name. This situation means the sensor actions on the most recently defined sensor with the same name take precedence.

For the scenario shown in `sensor.xml` in the following example:

- The first two sensors named `Service1` are identical. In addition, the configuration type for both is `serviceConfig` (composite sensors defined on a service binding component). Therefore, the sensors become one entry (the second one is ignored).
- The third sensor named `Service1` has a different configuration type of `eventConfig` (a composite sensor defined on a business event). Therefore, this sensor is represented with a separate entry.
- The two sensors named `PurchaseOrder Id` have different configuration types (`eventConfig` and `serviceConfig`). Therefore, they are represented with separate entries.
- The two sensors named `PurchaseOrder` have the same configuration type (`eventConfig`), but different expressions. Therefore, they are represented with separate entries.

```
<sensors xmlns="http://xmlns.oracle.com/bpel/sensor">
   <sensor sensorName="Service1" kind="service" target="undefined" filter="">
      <serviceConfig service="OrderPublisher_ep"
 expression="$in.property.tracking.ecid" operation="execute"
 outputDataType="string" outputNamespace="http://www.w3.org/2001/XMLSchema"/>
   </sensor>
   <sensor sensorName="Service1" kind="service" target="undefined" filter="">
      <serviceConfig service="OrderPublisher_ep"
 expression="$in.property.tracking.ecid" operation="execute"
 outputDataType="string" outputNamespace="http://www.w3.org/2001/XMLSchema"/>
   </sensor>
   <sensor sensorName="Service1" kind="event" target="undefined" filter=""
 xmlns:po="http://www.mycompany.com/ns/order">
      <eventConfig component="EventMediator"
 expression="$in/po:PurchaseOrder/po:OrderID"
 event="{http://mycompany.com/events/orders}OrderReceivedEvent"
 outputDataType="string" outputNamespace="http://www.w3.org/2001/XMLSchema"/>
   <sensor sensorName="Event1" kind="event" target="undefined" filter="">
```

```
      <eventConfig component="EventMediator" actionType="Subscribe"
 expression="$in.property.tracking.ecid"
 event="{http://mycompany.com/events/orders}OrderReceivedEvent"
 outputDataType="string" outputNamespace="http://www.w3.org/2001/XMLSchema"/>
   </sensor>
   <sensor sensorName="PurchaseOrder Id" kind="event" target="undefined" filter=""
 xmlns:po="http://www.mycompany.com/ns/order">
      <eventConfig component="EventMediator"
 expression="$in/po:PurchaseOrder/po:OrderID"
 event="{http://mycompany.com/events/orders}OrderReceivedEvent"
 outputDataType="string" outputNamespace="http://www.w3.org/2001/XMLSchema"/>
   </sensor>
  <sensor sensorName="PurchaseOrder Id" kind="service" target="undefined"
 filter="">
      <serviceConfig service="OrderPublisher_ep"
 expression="$in.property.tracking.ecid" operation="execute"
 outputDataType="string" outputNamespace="http://www.w3.org/2001/XMLSchema"/>
   </sensor>
   <sensor sensorName="PurchaseOrder" kind="event" target="undefined" filter=""
 xmlns:po="http://www.mycompany.com/ns/order">
      <eventConfig component="EventMediator" expression="$in/po:PurchaseOrder"
 event="{http://mycompany.com/events/orders}OrderReceivedEvent"
 outputDataType="PurchaseOrder"
 outputNamespace="http://mycompany.com/events/orders"/>
   </sensor>
   <sensor sensorName="PurchaseOrder" kind="event" target="undefined" filter=""
 xmlns:po="http://www.mycompany.com/ns/order">
      <eventConfig component="EventMediator"
 expression="$in/po:PurchaseOrder/po:OrderID"
 event="{http://mycompany.com/events/orders}OrderReceivedEvent"
 outputDataType="string" outputNamespace="http://www.w3.org/2001/XMLSchema"/>
   </sensor>
   </sensor>
</sensors>
```

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 on the Flow Instances page at the SOA Infrastructure, individual partition, and SOA composite application levels.

For more information about searching for composite sensors in Oracle Enterprise Manager Fusion Middleware Control, see Section "Monitoring and Deleting SOA Composite Application Instances at the SOA Infrastructure Level" and Section "Monitoring and Deleting SOA Composite Application Instances from the Application Home Page" of *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

Creating and Managing Composite Sensors During Runtime from Oracle SOA Composer

You can create, update, and delete composite sensors during runtime from Oracle SOA Composer without having to redeploy a SOA composite application. The following example describes how to create a composite sensor. Changes to composite sensors can be carried to new revisions of the composite through patching.

Ensure that you understand the issues around using duplicate names for composite sensors. For more information, see What You May Need to Know About Duplicate Composite Sensor Names.

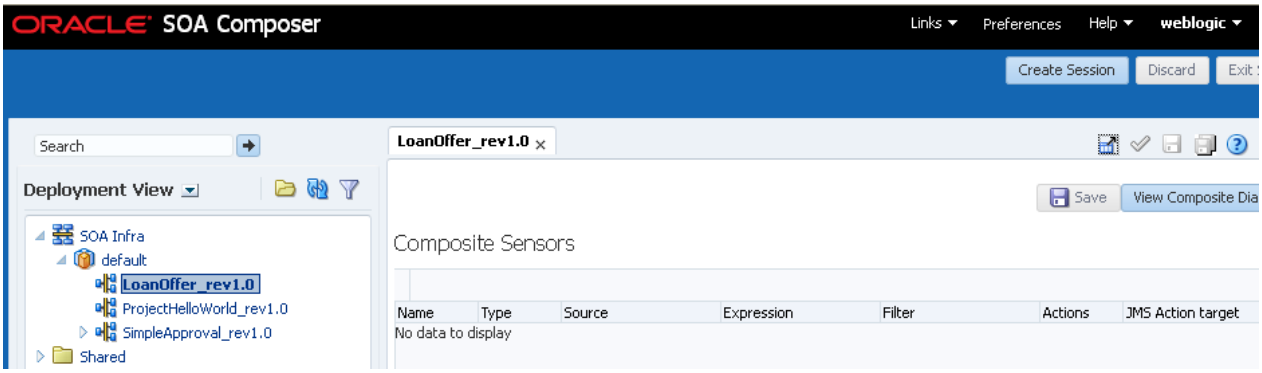
To create and manage composite sensors during runtime from Oracle SOA Composer:

1. Log in to Oracle SOA Composer.

   `http://host:soa_server_port/soa/composer`

2. Expand the navigator on the left and double-click the composite in which to make changes. Figure 53-12 provides details.

Figure 53-12 Oracle SOA Composer



3. Click **Create Session**.

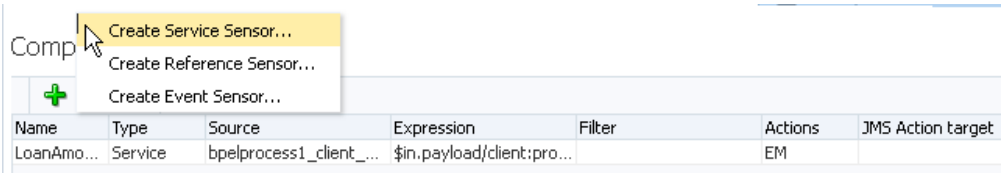
   The page is refreshed to display the **Add**, **Edit**, and **Delete** icons.

4. Click the **Add** icon and select an option:

   - **Create Service Sensor**: Data is coming from a service binding component call.
   - **Create Reference Sensor**: Data is coming from a reference binding component call.
   - **Create Event Sensor**: Data is coming from a service component that has subscribed to a business event.

   For this example, **Create Service Sensor** is selected because the data is coming from a service binding component call. Figure 53-13 provides details.

Figure 53-13 Composite Sensor Creation



   The Create Composite Sensor dialog is displayed.

5. Click the **Edit** icon in the **Expression** section, and select an option:

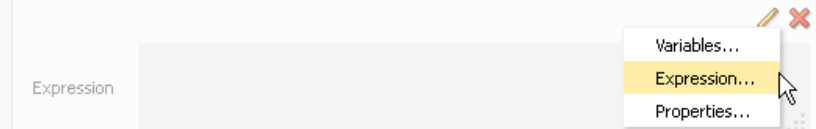
   - **Variables**: Select to create an expression value for a variable.

- **Expression**: Select to invoke the Expression Builder dialog for creating an XPath expression. This action always captures values as strings.
- **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 scope activity (in BPEL 2.0).

For this example, **Expression** is selected to build an XPath expression.

Figure 53-14 provides details.

Figure 53-14 XPath Expression Selection of Create Composite Sensor Dialog



The selections of variables, expressions, and header properties are the same as with the Create Composite Sensor dialog in Oracle JDeveloper, as described in Table 53-1.

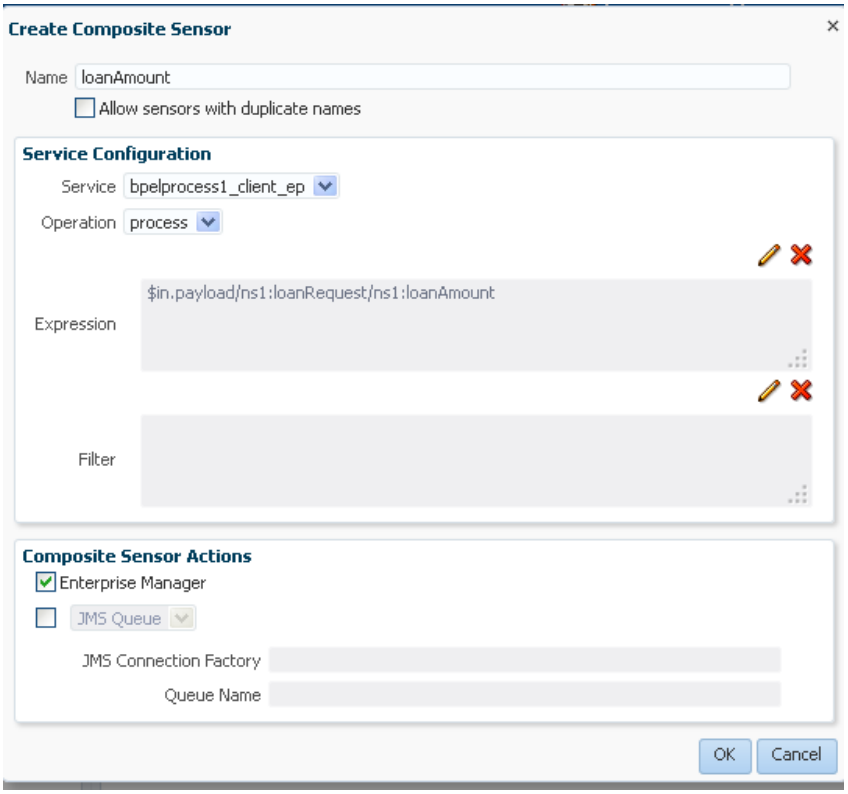
The Expression Builder dialog is displayed.

6. Build an XPath expression and click **OK**. You can also select custom XPath expressions that you created.

   You are returned to the Create Composite Sensor dialog.

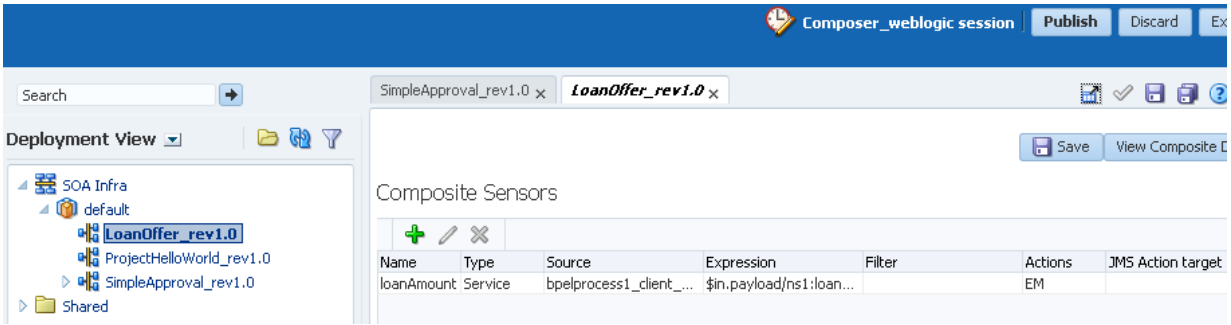
7. Select the **Enterprise Manager** check box in Figure 53-15 to make this composite sensor a searchable, trackable field from the Flow Instances page of a SOA composite application in Oracle Enterprise Manager Fusion Middleware Control, and click **OK**. If you do not select this check box, the composite sensor is not searchable.

Figure 53-15 Create Composite Sensor



The new composite sensor is displayed, including the sensor name, the type and name of the component in which the sensor is defined, any XPath expression or filter defined on the sensor, the storage location for runtime sensor data (Enterprise Manager or a JMS queue and topic), and any JMS targets. Figure 53-16 provides details.

Figure 53-16 Composite Sensors in Oracle SOA Composer



8. Click **Save**.

9. In the upper right corner, click **Publish** to publish this session. Figure 53-17 provides details.

Figure 53-17 Publish Button

10. Enter an optional description for the session when prompted, then click **OK**.

    The composite sensor is now running automatically in the deployed SOA composite application.

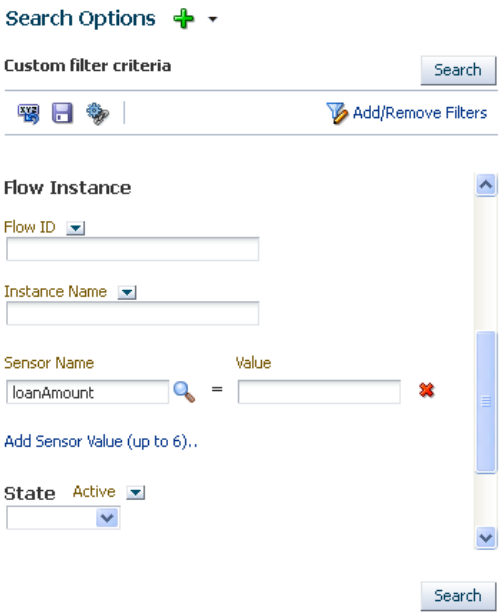
11. Go to the Test Web Service page in Oracle Enterprise Manager Fusion Middleware Control to invoke a new instance. For information about the Test Web Service page, see "Initiating a SOA Composite Application Test Instance" of *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

12. Create a new instance of the SOA composite application that includes the composite sensor (for this example, named **loanAmount**), and click **Invoke**.

13. Go to the Flow Instances page of the SOA Infrastructure.

14. In the **Sensor Name** field of the **Flow Instance** part of the **Search Options** section, specify the composite sensor you added. Figure 53-18 provides details.

    Figure 53-18 Searchable Field

15. Click **Search**.

16. In the **Search Results** table, select the instance of the SOA composite sensor and click **Show Details**.

    Instance details are displayed in the **Faults**, **Composite Sensor Values**, **Composites**, and **Resequencing Groups** tabs at the bottom of the page.

17. Click the **Composite Sensor Values** tab.

    This tab displays the values of composite sensors detected in the selected business flow.

    - **Name**: Displays the composite sensor name (for this example, **loanAmount**).
    - **Value**: Displays the value assigned to the composite sensor.

- **Location**: Displays the service or reference binding component or service component in which the composite sensor is defined.
- **Composite**: Displays the SOA composite application in which the composite sensor is defined.

18. If you want to edit or delete the composite sensor, return to Oracle SOA Composer, as shown in Figure 53-16, and click **Create Session**.

    The page is refreshed to again display the **Add**, **Edit**, and **Delete** icons.

19. If you set the `oracle.soacomposer.composite.showSensorXmlFiles` Oracle WebLogic Server startup script system property, the **Show Sensor XML** button appears at the bottom of the page.

20. Click this property to show `sensor.xml` and `sensor-action.xml` content. This helps you to test both to see that they are what you expect them to be.

    If you later import this SOA composite application in to Oracle JDeveloper, the composite sensors created during runtime in Oracle SOA Composer are displayed.

What You May Need to Know About Viewing Composite Sensor Changes in Oracle SOA Composer

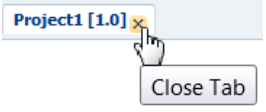
When you add or remove composite sensors in Oracle SOA Composer, you must close and reopen the project tab above the Composite Sensors table to see the changes. For example:

1. Create and deploy a SOA composite application with a composite sensor (for this example, named p1).
2. Log in to Oracle SOA Composer, and select the composite in the navigator.

   The p1 composite sensor is displayed.
3. Create an additional composite sensor (for this example, named p2) in the composite and redeploy it.
4. In the navigator tree of Oracle SOA Composer, click the **Refresh** button, and select the composite.

   Only composite sensor p1 is displayed, and not p2.
5. Close the project tab above the Composite Sensors page, as shown in Figure 53-19, and reopen it by selecting the composite in the navigator.

Figure 53-19 Composite Tab in Composite Sensors Page



This enables composite sensors p1 and p2 to be displayed.

54

Creating Dynamic Business Processes

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.

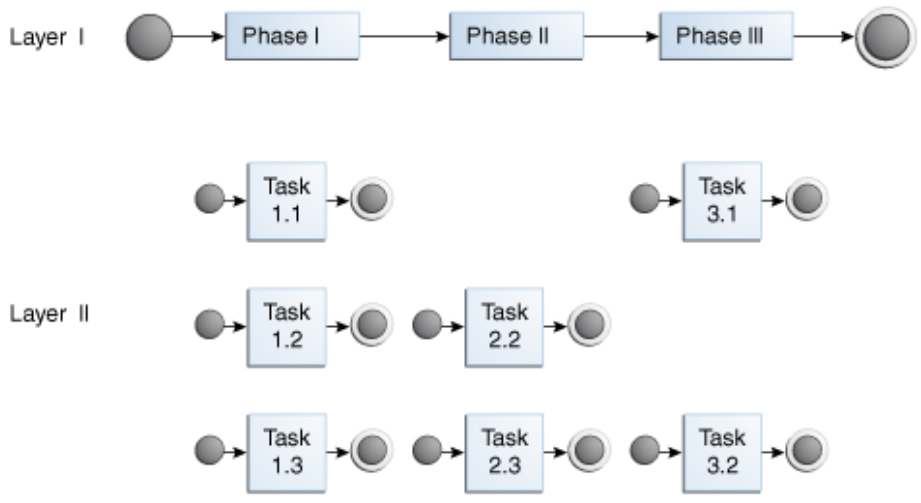
To illustrate further, assume you have an application that performs multichannel banking using various processes. In this scenario, the execution of each process depends on the channel for each particular process instance.

This chapter includes the following sections:

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 54-1 illustrates this behavior.

Figure 54-1 Two-Layer BPM



In Figure 54-1, 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:

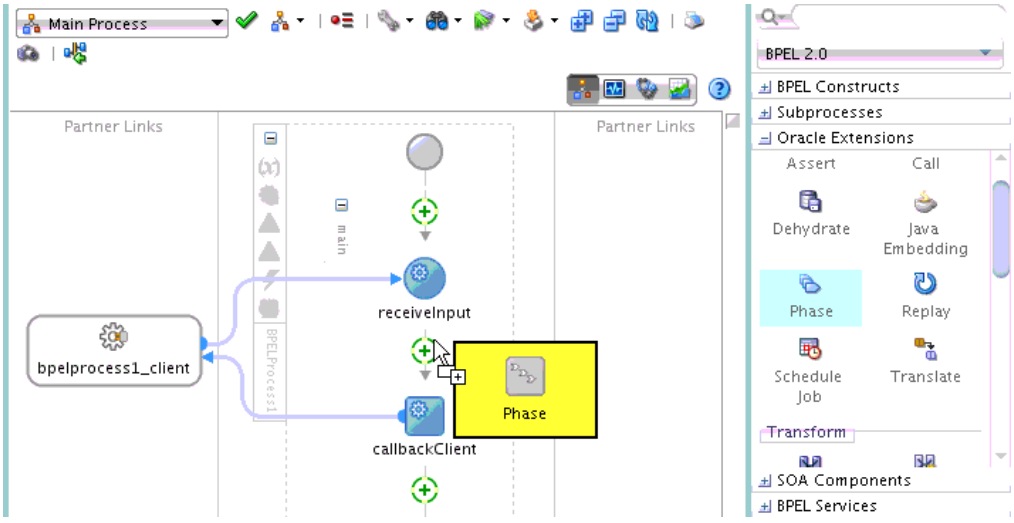
- *Designing Business Rules with Oracle Business Process Management*
- *Java API Reference for Oracle Business Rules*

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 activity to a process declaratively in Oracle BPEL Designer by dragging and dropping it from the **Oracle Extensions** section of the Components window to the process model. Figure 54-2 provides details.

Figure 54-2 Phase Activity in Oracle 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.

How to Create a Phase Activity

You create the phase activity for your composite application after you have created the necessary variables.

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 an existing variable or select the **Variables** folder and click the **Add** icon to create a new variable.
6. Click **OK**. The Phase dialog is displayed with the variable populated.
7. From the **Input and Output Variables** icon, select **Add Output Variable**. The dialog for selecting a variable appears.Select an existing variable or select the **Variables** folder and click the **Add** icon to create a new variable.
8. Click **OK**. The Phase dialog is displayed with the input and output variable names populated.Click **OK**. The Oracle BPEL Designer displays the BPEL process.
9. From the **File** menu, select **Save All**.
10. Close the BPEL process.
11. Click the *composite_name* link (that is, the `composite.xml` file) above Oracle BPEL Designer. The SOA Composite Editor appears.

What Happens When You Create a Phase Activity

When you create a phase activity, the artifacts described in Table 54-1 are created.

Table 54-1 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.

Table 54-1 (Cont.) Artifacts Created with a Phase Activity

Artifact	Description
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.
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 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 Oracle Mediator to evaluate the decision table.

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.

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.

Note:

No transformation, assignment, or validation can be performed on a payload.

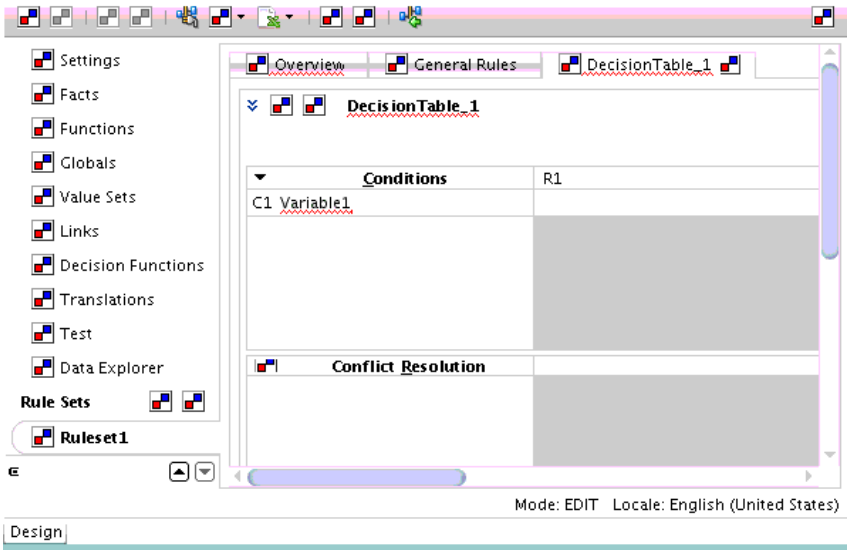
Creating the Dynamic Routing Decision Table

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.

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 54-3 shows a sample decision table within the Oracle Business Rules Designer.

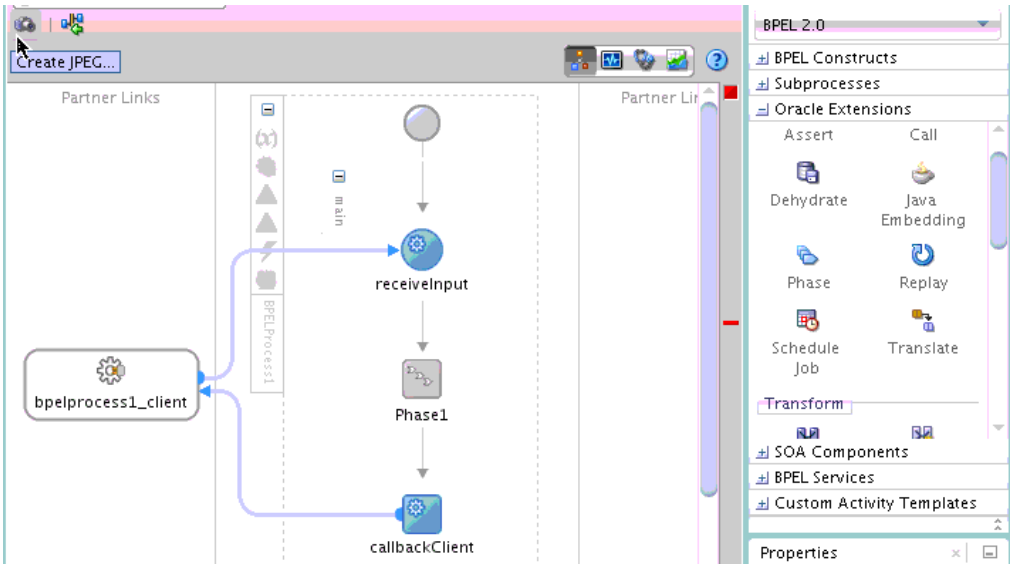
Figure 54-3 Sample Decision Table



You can leave the information 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 shown in Figure 54-4.

Figure 54-4 Completed Level-1 Phase in Oracle JDeveloper



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 Oracle Mediator. 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.

55

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. It also describes using Java Architecture for XML Binding (JAXB) and the EclipseLink O/X-Mapper (OXM) to map Java classes to XML data.

This chapter includes the following sections:

For more information about the WebLogic SCA functionality used by Oracle SOA Suite, see *Developing WebLogic SCA Applications for Oracle WebLogic Server*.

For samples about how to use the spring framework, see the Oracle SOA Suite samples site.

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

As with all service components, spring components are defined in the `composite.xml` file. The spring component defined in the `composite.xml` file has service and reference elements with `binding.java`.

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 (EJB) with SOA composite applications through use of Java interfaces (with no requirement for SDO parameters). For information, see Integrating Enterprise JavaBeans with Composite Applications .

Integration of Java and WSDL-Based Components in the Same SOA Composite Application

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 EJB, 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, EJB 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.

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.

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 in **Design** view to create and modify the `composite.xml` 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 the following example:

```
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 an external web service, or an EJB. The following example provides details:

```
public interface StockQuote {
  public double getQuote (String symbol);
}
```

The `composite.xml` file for the spring framework lists the `PortfolioService` service and the `StockQuote` service with the `interface.java` definitions. The following example provides details.

```
<component name="PortfolioComp">
    <implementation.spring src="Spring/PortfolioComp.xml"/>
    <componentType>
      <service name="PortfolioService">
        <interface.java interface="com.bigbank.PortfolioService"/>
      </service>
      <reference name="StockService">
        <interface.java interface="com.bigbank.StockQuote"/>
      </reference>
    </componentType>
  </component>
```

The implementation class implements the service interface and provides a setter for the reference interface. The following example provides details:

```
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. The following example provides details:

```
<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>
```

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 the following example in the spring context file with the spring editor in Oracle JDeveloper:

```
<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 as shown in the following example:

```
    <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`).

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:

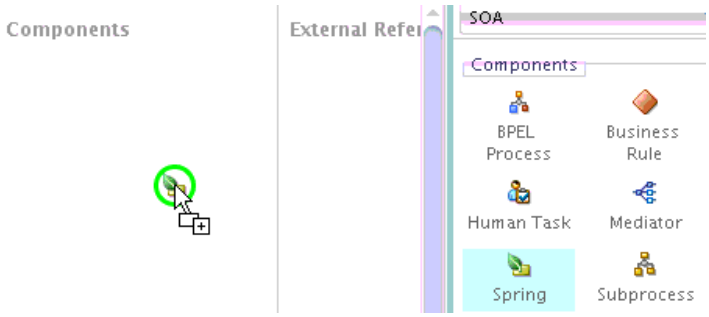
- To Java interface-based EJB services and references (Java-to-Java integration)
- To an Oracle Mediator service component (Java-to-WSDL integration)

How to Create a Spring Service Component in Oracle JDeveloper

To create a spring service component in Oracle JDeveloper:

1. From the Components window, drag a **Spring** service component into the , as shown in Figure 55-1.

   Figure 55-1 Spring Context Service Component

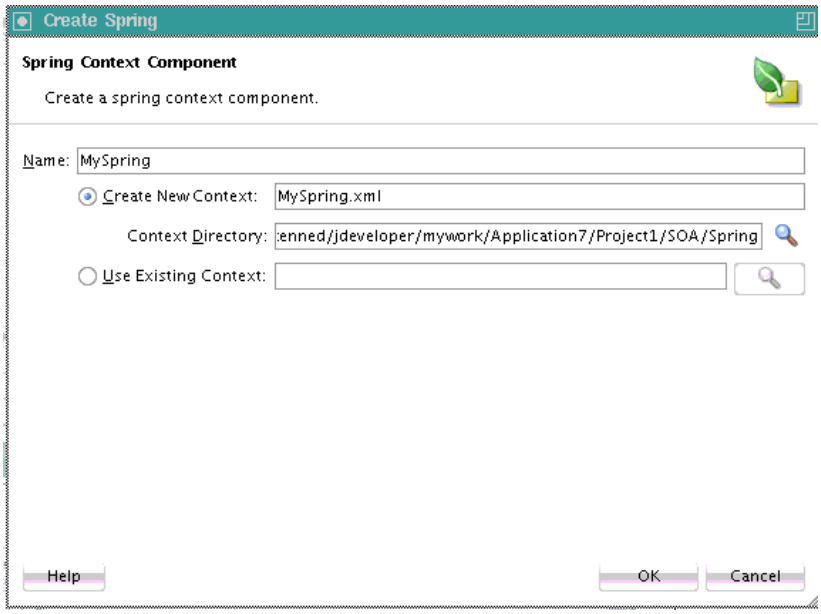
   

   The Create Spring dialog is displayed.

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 55-2 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 55-2 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 Components window while inside the spring editor.

3. Click **OK**.

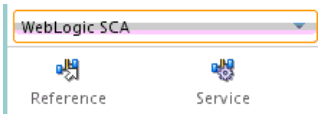
   A spring icon is displayed in the SOA Composite Editor.

4. If the contents are not automatically displayed, double-click the icon to display the contents of the spring context in the spring editor.

5. From the Components window, select **Weblogic SCA** from the dropdown list.

   The list is refreshed to display the selections shown in Figure 55-3.

Figure 55-3 WebLogic SCA Menu



6. Drag a **Service** icon into the spring editor.

   The Insert Service dialog appears.

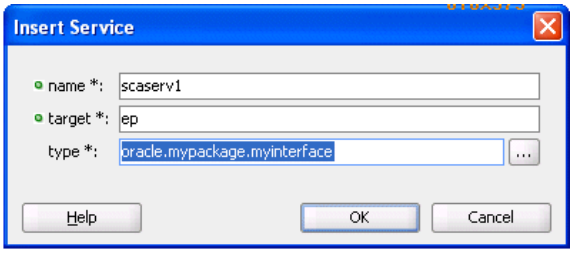
7. Complete the fields shown in Table 55-1 to define the target bean and Java interface.

Table 55-1 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.
type	Enter the Java interface.

When complete, the Insert Service dialog looks as shown in Figure 55-4.

Figure 55-4 Insert Service Dialog



8. Click **OK**.

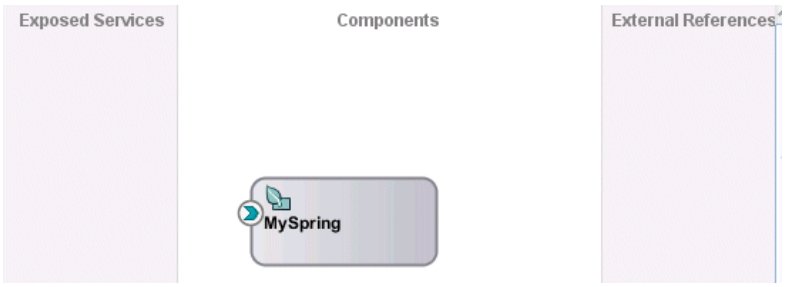
The target bean becomes the service interface in the spring context.

```
<?xml version="1.0" encoding="UTF-8" ?>
<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.xsd
 http://www.springframework.org/schema/tool
 http://www.springframework.org/schema/tool/spring-tool.xsd
 http://www.springframework.org/schema/util
 http://www.springframework.org/schema/util/spring-util.xsd
 http://www.springframework.org/schema/aop
 http://www.springframework.org/schema/aop/spring-aop.xsd
 http://www.springframework.org/schema/cache
 http://www.springframework.org/schema/cache/spring-cache.xsd
 http://www.springframework.org/schema/context
 http://www.springframework.org/schema/context/spring-context.xsd
 http://www.springframework.org/schema/task
 http://www.springframework.org/schema/task/spring-task.xsd
 http://www.springframework.org/schema/jee
 http://www.springframework.org/schema/jee/spring-jee.xsd
 http://www.springframework.org/schema/lang
 http://www.springframework.org/schema/lang/spring-lang.xsd
 http://www.springframework.org/schema/tx
 http://www.springframework.org/schema/tx/spring-tx.xsd
 http://www.springframework.org/schema/jdbc
```

```
 http://www.springframework.org/schema/jdbc/spring-jdbc.xsd
 http://www.springframework.org/schema/jms
 http://www.springframework.org/schema/jms/spring-jms.xsd
 http://www.springframework.org/schema/oxm
 http://www.springframework.org/schema/oxm/spring-oxm.xsd
 http://www.springframework.org/schema/mvc
 http://www.springframework.org/schema/mvc/spring-mvc.xsd
 http://xmlns.oracle.com/weblogic/weblogic-sca META-INF/weblogic-sca.xsd">
  <!--Spring Bean definitions go here-->
  <sca:service name="scaserv1" target="cp"
 type="oracle.mypackage.myinterface"/>
</beans>
```

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 55-5.

Figure 55-5 Service Handle



9. Return to the spring editor.
10. Drag a **Reference** icon from the list shown in Figure 55-3 into the spring editor.

    The Insert Reference dialog is displayed.
11. Complete the dialog, as shown in Table 55-2, and click **OK**.

Table 55-2 Insert Reference Dialog

Field	Description
name	Enter a name.
type	Enter the Java interface.

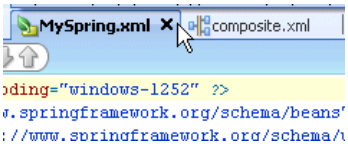
When complete, the spring context displays the service and reference in the spring editor.

```
<?xml version="1.0" encoding="UTF-8" ?>
<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.xsd
http://www.springframework.org/schema/tool
```

```
http://www.springframework.org/schema/tool/spring-tool.xsd
http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop.xsd
http://www.springframework.org/schema/cache
http://www.springframework.org/schema/cache/spring-cache.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd
http://www.springframework.org/schema/task
http://www.springframework.org/schema/task/spring-task.xsd
http://www.springframework.org/schema/jee
http://www.springframework.org/schema/jee/spring-jee.xsd
http://www.springframework.org/schema/lang
http://www.springframework.org/schema/lang/spring-lang.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx.xsd
http://www.springframework.org/schema/jdbc
http://www.springframework.org/schema/jdbc/spring-jdbc.xsd
http://www.springframework.org/schema/jms
http://www.springframework.org/schema/jms/spring-jms.xsd
http://www.springframework.org/schema/oxm
http://www.springframework.org/schema/oxm/spring-oxm.xsd
http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc.xsd
http://xmlns.oracle.com/weblogic/weblogic-sca META-INF/weblogic-sca.xsd">
 <!--Spring Bean definitions go here-->
 <sca:service name="scaserv1" target="cp"
type="oracle.mypackage.myinterface"/>
 <sca:reference name="scaref1" type="external.bean.myInterface"/>
</beans>
```

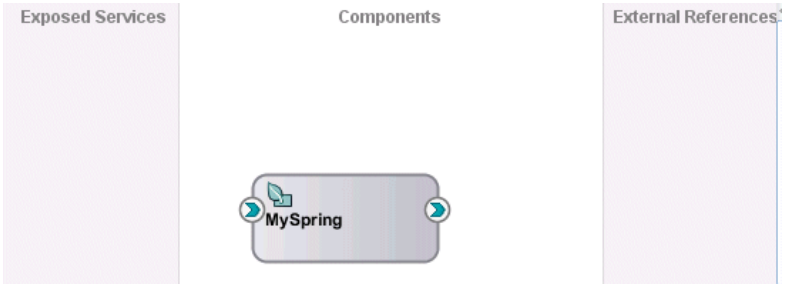
12. Close the spring context file, as shown in Figure 55-6.

Figure 55-6 Spring Context File



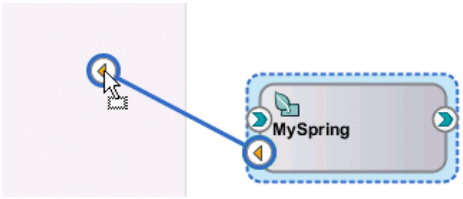
A handle is added to the right side of the spring service component, as shown in Figure 55-7.

Figure 55-7 Reference Handle



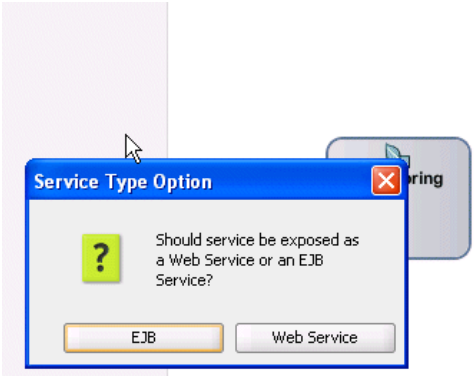
13. Drag the left handle into the **Exposed Services** swimlane to create a service binding component, as shown in Figure 55-8.

Figure 55-8 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 55-9.

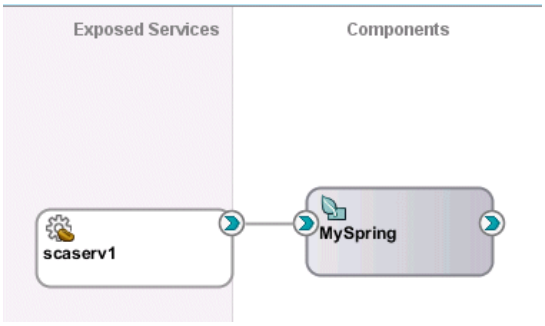
Figure 55-9 Service Type To Create



- **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.

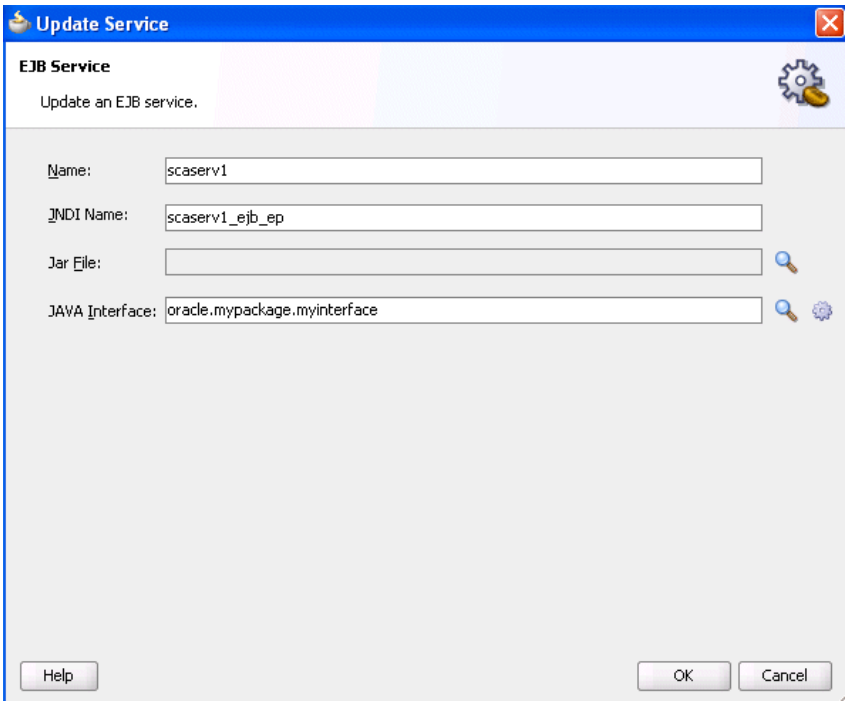
14. Select to expose this service as either an EJB or web service. A service is automatically created in the **Exposed Services** swimlane and wired to the spring service component (for this example, **EJB** is selected). Figure 55-10 provides details.

Figure 55-10 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 55-11. The configuration details were created from the values you entered in the Insert Service dialog in Step 7.

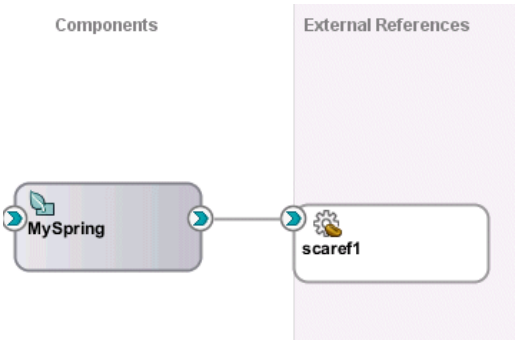
Figure 55-11 EJB Service Dialog in Exposed Services Swimlane



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.

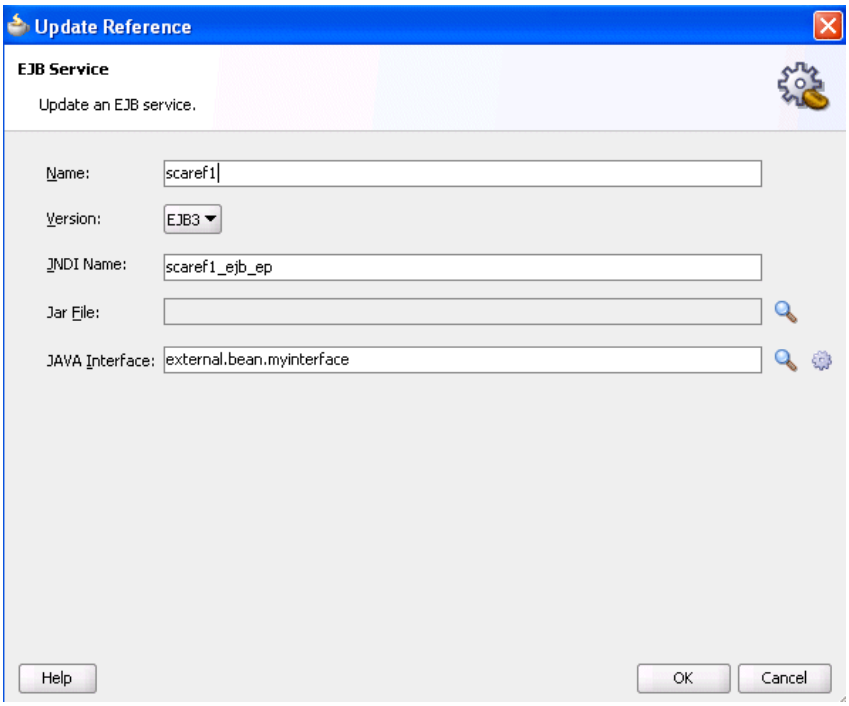
    You are prompted with the same spring type option message as shown in Step 13.
19. Select an option to expose this reference. A reference is automatically created in the **External References** swimlane and wired to the spring service component (for this example, **EJB** is selected). Figure 55-12 provides details.

Figure 55-12 EJB Reference Binding Component Wired to the Spring Service Component



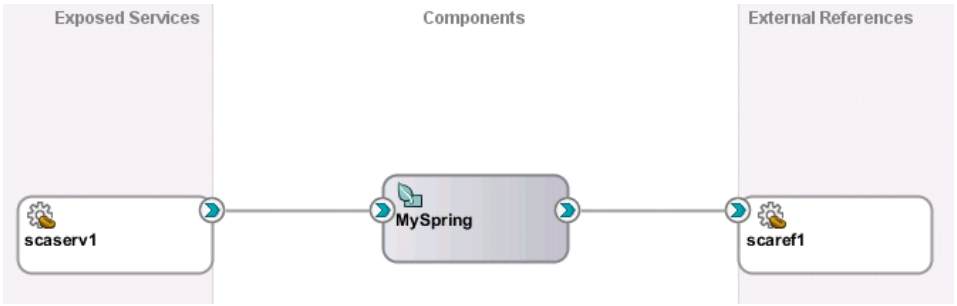
20. Double-click the EJB reference to display the automatically completed configuration, as shown in Figure 55-13. The configuration details were created from the values you entered in the Insert Reference dialog in Step 11.

Figure 55-13 EJB Reference Dialog in External References Swimlane



21. Close the dialog and return to the SOA Composite Editor, as shown in Figure 55-14.

Figure 55-14 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 55-15) and the left handle of the spring service component (as shown in Figure 55-16). The Java interface is displayed.

Figure 55-15 Java Interface of Service

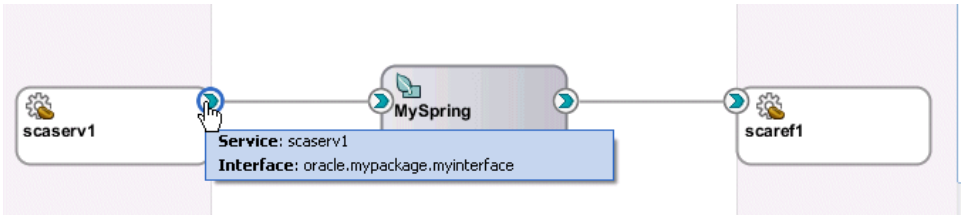
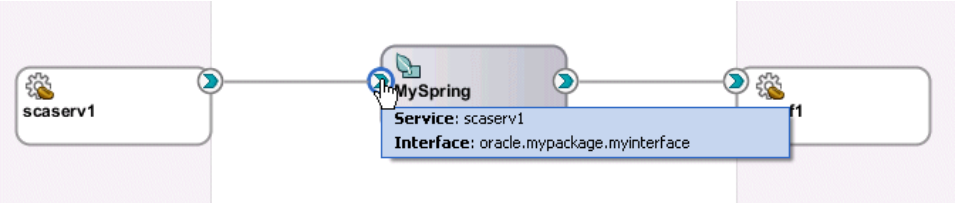


Figure 55-16 Java Interface of Spring Service Component



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.

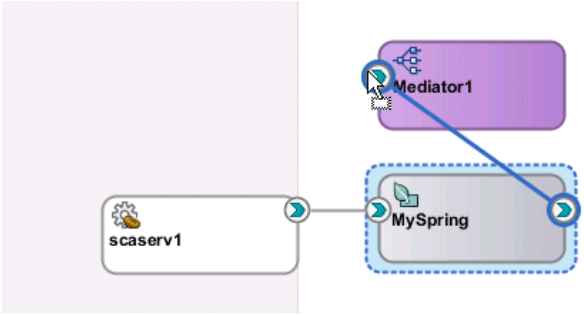
24. Select **Source** view for the `composite.xml` file to display similar details.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- Generated by Oracle SOA Modeler version 12.1.3.0.0 at [5/16/14 3:05 AM].
 -->
<composite name="Project1"
. . .
. . .
<service name="scaserv1">
  <interface.java interface="oracle.mypackage.myinterface"/>
  <binding.ejb uri="scaserv1_ejb_ep" ejb-version="EJB3"/>
</service>
<property name="productVersion" type="xs:string"
 many="false">12.1.3.0.0</property>
<property name="compositeID" type="xs:string"
 many="false">4c07dbf0-5c01-450e-bde6-8c3866f45edc</property>
<component name="MySpring">
  <implementation.spring src="Spring/MySpring.xml"/>
  <componentType>
    <service name="scaserv1">
      <interface.java interface="oracle.mypackage.myinterface"/>
    </service>
    <reference name="scaref1">
      <interface.java interface="external.bean.myInterface"/>
    </reference>
  </componentType>
</component>
<reference name="scaref1">
  <interface.java interface="external.bean.myInterface"/>
  <binding.ejb uri="scaref1_ejb_ep" ejb-version="EJB3"/>
</reference>
<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>
```

25. 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 Java interface is generated from the Oracle Mediator's existing WSDL interface. The following steps provide details.

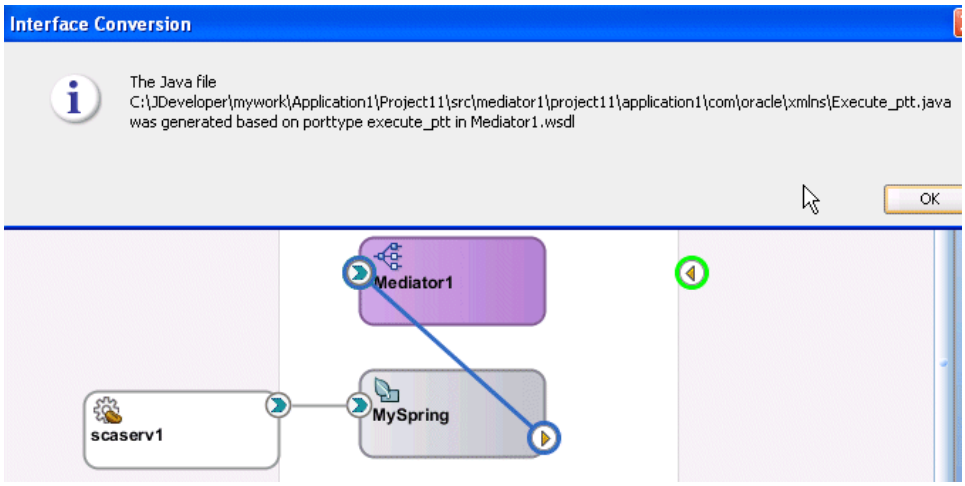
    a. Drag the right handle of the spring service component to the Oracle Mediator, as shown in Figure 55-17.

Figure 55-17 Integration of Spring Service Component and Oracle Mediator



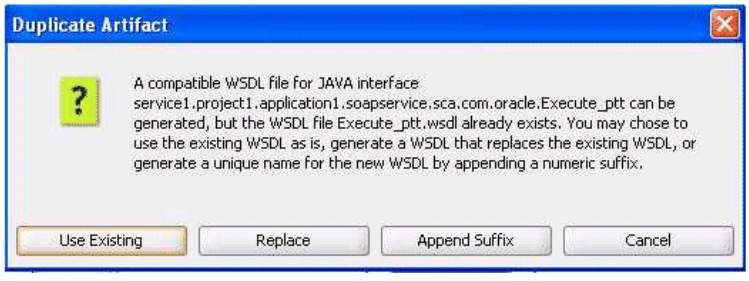
b. Click **OK** when prompted to acknowledge that a compatible interface was created from the Oracle Mediator WSDL file.

Figure 55-18 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 55-19 provides details.

Figure 55-19 Existing WSDL File



c. Place the cursor over both the right handle of the spring service component (as shown in Figure 55-20) and the left handle of the Oracle Mediator (as shown in Figure 55-21) to display the compatible interface.

Figure 55-20 Spring Service Component Interface

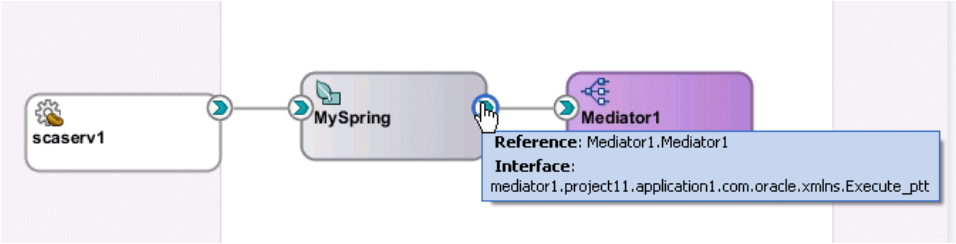
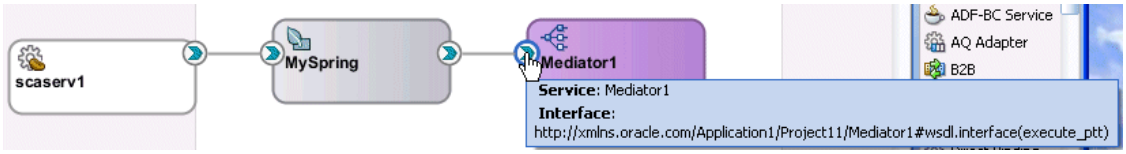


Figure 55-21 Oracle Mediator Interface



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>
```

Note:

- 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 (including the JAR files in the `SCA-INF/lib` directory), but the source file does exist in the SOA project, you are prompted to automatically compile the source.
- You can also create BPEL process partner links with services that use Java interfaces. You select this type of service in the Service Explorer dialog when creating a partner link. For more information, see Introduction to Partner Links.

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 is not reloaded.

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.

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.

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 flow 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 How to Reference Predefined Spring Beans in the Spring Context File.

IHeaderHelperBean.java Interface for headerHelperBean

The following example shows the `IHeaderHelperBean.java` interface for the `headerHelperBean` bean:

```
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/>
 * 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);
}
```

IInstanceHelperBean.java Interface for instancerHelperBean

The following example shows the `IInstanceHelperBean.java` interface for the `instanceHelperBean` bean:

```
package oracle.soa.platform.component.spring.beans;

import oracle.integration.platform.instance.engine.ComponentInstanceContext;
/**
 * Instancehelper Bean, gives access to composite / component + instance
 information
 * <br/>
 * 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 ();

}
```

ILoggerBean.java Interface for loggerBean

The following example shows the `ILoggerBean.java` interface for the `loggerBean` bean:

```
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/>
 * To use this bean from within your context, declare property
 * with ref="loggerBean". E.g.
 * &lt;property name="logger" ref="<b>loggerBean</b>"/>
 */
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);

}
```

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>
. . .
. . .
```

JAXB and OXM Support

Oracle Fusion Middleware provides support for using JAXB and EclipseLink 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 an EJB with SOA composite applications. For more information, see Integrating Enterprise JavaBeans with Composite Applications .

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.

The EXM file name must match the Java class name and reside in the same package location. For example, if the Java class is named `pack1.pack2.myJavaInterface.class`, the EXM file must be named `pack1/pack2/myJavaInterface.exm`.

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 Applications window 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.

If you place the EXM (and OXM) files in `SCA-INF/src`, ensure that your Oracle JDeveloper project is configured so that `SCA-INF/src` is the default source directory (right-click the project name, and select **Project Properties** > **Java Source Paths**). EXM files can also be found in JAR files that are in the project's class path.

When you drag and drop a Java interface (Enterprise JavaBeans) to a BPEL process, Oracle SOA Suite checks to see if the EXM file exists. If it does, it is passed to the web services `java2wsdl` API.

After the WSDL file is generated, an informational message is displayed. If an EXM file was used, the message displayed takes the following format:

```
The WSDL file {0} was generated based on the JAVA class {1} using extended mapping
file {2}
```

The following provides an example of an EXM file:

```
<java-wsdl-mapping name="com.hello.sei.MyServiceEndpointInterface"
          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="com.hello.foo"  file-path="./foo-oxm.xml"/>
      <toplink-oxm      java-package="com.hello.coo">
           <xml-bindings
xmlns="http://www.eclipse.org/eclipselink/xsds/persistence/oxm">
               <xml-schema
                   element-form-default="QUALIFIED"
                   attribute-form-default="UNQUALIFIED"
                   namespace="urn:customer">
                   <xml-ns prefix="ns1" namespace-uri="urn:customer" />
               </xml-schema>
               <java-types>
                   <java-type name="Person" xml-transient="true">
```

```
                    <java-attributes>
                        <xml-transient java-attribute="id"/>
                    </java-attributes>
                </java-type>
                <java-type name="Customer">
                    <xml-see-also>org.example.employee.Employee</xml-see-also>
                </java-type>
            </java-types>
        </xml-bindings>
    </toplink-oxm>
  </xml-schema-mapping>
. . .
</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-181 that override or define attributes on the service endpoint interface (SEI) and JAXB annotations for the value types used 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
```

Configuring Groovy and Aspectj Classes with the Spring Service Component

If you configure a Groovy or Aspectj class in the spring configuration file, you must follow these conventions:

- Use the `classpath` protocol:

  ```
  script-source="classpath:"
  ```

  Using a relative file path is not possible because the SCA package is not treated as a regular JAR file for the class loader. For example, the following `classpath` protocol indicates to find the Groovy file from the class path.

  ```
  script-source="classpath:service/GroovyGreeter.groovy"
  ```

- Add Groovy and Aspectj files in any of the following directories when using the `classpath` protocol. No other directories are possible.
  - `SCA-INF/classes`
  - `SCA-INF/lib`

  - Shared SOA `lib`

    If your build scripts are configured to clean the `classes` directory, either put the Groovy files in the `SCA-INF/lib` directory or design your build scripts to prevent cleaning.

- Add spring extension JAR file libraries for Groovy or Aspectj to the class path of the managed server's `setDomainENV.sh` or `setDomainENV.bat` file and restart the server. This ensures that deployment is successful. The restart is required because spring uses Java reflection to instantiate aspect-oriented programming (AOP). The use of reflection restricts the search for classes to the system class loader. Any changes to the system class loader require a server restart.

Troubleshooting Spring Errors

This section describes how to troubleshoot errors with the spring service component.

Spring Bean Interface to Invoke Cannot Be Found

Assume you have a SOA composite application in which a BPEL process invokes a spring context. However, the spring bean interface to invoke cannot be found. The administration server diagnostic log file displays the error shown in the following example:

```
[2012-04-09T10:30:07.499-07:00] [AdminServer] [NOTIFICATION] [SOA-31704]
[oracle.integration.platform.blocks.java] [tid: [ACTIVE].ExecuteThread: '2' for
queue: 'weblogic.kernel.Default (self-tuning)'] [userId: <anonymous>] [ecid:
11d1def534ea1be0:2058db3f:1369787a1b8:-8000-0000000000002be6,0:2] [WEBSERVICE_
PORT.name: SOACohSpringBPELProcess_pt] [APP: soa-infra] [composite_name:
SOACohSpringProj] [component_name: SOACohSpringBPELProcess] [component_instance_
id: 270006] [J2EE_MODULE.name: fabric] [WEBSERVICE.name: soacohspringbpelprocess_
client_ep] [J2EE_APP.name: soa-infra] No mapping found for class
SOACohSpringProj.CohEJBInterface.
```

Ensure that you deploy the JAR file containing the class into the `SCA-INF/lib` directory or the classes into the `SCA-INF/classes` directory of the SAR file.

Unable to Add a Spring Service Component in the SOA Composite Editor

The Oracle SOA Suite Quick Start installation automatically includes the spring extension files for invoking the spring editor. This enables you to successfully add a spring service component in the SOA Composite Editor and invoke the Create Spring dialog, as described in How to Create a Spring Service Component in .

If you use the standard Oracle JDeveloper installation outside of Oracle SOA Suite, you must install the spring editor by selecting **Check for Updates** from the **Help** main menu in Oracle JDeveloper, then selecting the spring extension files in the Update Center. Otherwise, you cannot successfully add a spring service component into the SOA Composite Editor and invoke the Create Spring dialog. Instead, you receive the error shown in Figure 55-22.

Figure 55-22 Spring Unavailability Error

Part X

Appendices

This part describes Oracle SOA Suite appendixes.

This part contains the following appendixes:

A

BPEL Process Activities and Services

This appendix describes the BPEL process activities and services that you use when designing a BPEL process in a SOA composite application.

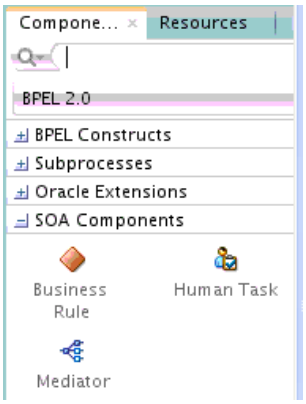
This appendix includes the following sections:

- Introduction to Activities and Components
- Introduction to BPEL 1.1 and 2.0 Activities
- Introduction to BPEL Services

Introduction to Activities and Components

When you expand **SOA Components** in the Components window of Oracle BPEL Designer, service components are displayed. Figure A-1 shows the service components that display for a BPEL 2.0 or 1.1 process.

Figure A-1 SOA Components



See the following sections for additional details about service components.

- BPEL process

  See Using the BPEL Process Service Component

- Oracle Mediator

  See Using the Oracle Mediator Service Component

- Business rule

  See Using the Business Rules Service Component

- Human task

  Using the Human Workflow Service Component

- Spring

  Integrating the Spring Framework in SOA Composite Applications

For information about Oracle BPEL Designer, see Getting Started with Oracle BPEL Process Manager .

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 can be added to 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 Components window of Oracle BPEL Designer. The activities display in the following 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.
- **Subprocesses**: Displays any created subprocesses. If no subprocesses have been created, this category is empty. For more information about subprocesses, see Introduction to Standalone and Inline BPEL Subprocess Invocations.
- **Oracle Extensions**: Displays extension activities that add value and ease of use to BPEL 1.1 and 2.0 functionality.
- **SOA Components**: Displays the business rules, human tasks, and Oracle Mediator service components that can be added to a BPEL process.
- **BPEL Services**: Displays the partner links that can be added to a BPEL process, including JCA adapters (AQ, file, FTP, database, JMS, MQ, Oracle User Messaging Service, socket, JDE World, SAP, LDAP server, Coherence cache, and third-party), Oracle BAM 11*g* binding component, Oracle Healthcare binding component, Oracle B2B binding component, EJB binding component, ADF-BC binding component, Oracle E-Business Suite adapter, direct binding component, HTTP binding component, and Oracle Managed File Transfer (MFT) adapter.
- **Custom Activity Templates**: Displays any created custom scope activity templates. For more information about templates, see Introduction to Templates.

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	Assign Activity
Assert	Oracle Extensions	Yes	Yes	Assert 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
Bind Entity	Oracle Extensions	Yes	No	Bind Entity Activity
Call	Oracle Extensions	No	Yes	Call Activity
Compensate	BPEL Constructs	Yes	Yes	Compensate Activity
CompensateScope	BPEL Constructs	No	Yes	CompensateScope Activity
Create Entity	Oracle Extensions	Yes	No	Create Entity Activity
Dehydrate	Oracle Extensions	Yes	Yes	Dehydrate Activity
Dynamic Partner Link	BPEL Constructs	Yes	No	Dynamic Partner Link Activity
Email	Oracle Extensions	Yes	Yes	Email Activity
Empty	BPEL Constructs	Yes	Yes	Empty Activity
Exit	BPEL Constructs	No	Yes **Note:** Replaces the terminate activity in BPEL 2.0.	Exit Activity
Flow	BPEL Constructs	Yes	Yes	Flow Activity
FlowN	Oracle Extensions	Yes	No **Note**: Replaced by the forEach activity in BPEL 2.0	FlowN Activity
forEach	BPEL Constructs	No	Yes **Note:** Replaces the FlowN activity in BPEL 2.0.	forEach Activity
If	BPEL Constructs	No	Yes **Note**: Replaces the switch activity in BPEL 2.0.	If Activity
IM	Oracle Extensions	Yes	Yes	IM Activity
Invoke	BPEL Constructs	Yes	Yes	Invoke Activity
Java Embedding	Oracle Extensions	Yes	Yes	Java Embedding 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
Partner Link	BPEL Constructs	Yes	Yes	Partner Link Activity
Phase	Oracle Extensions	Yes	Yes	Phase Activity
Pick	BPEL Constructs	Yes	Yes	Pick Activity
Receive	BPEL Constructs	Yes	Yes	Receive Activity
Receive Signal	Oracle Extensions	Yes	Yes	Receive Signal Activity
Remove Entity	Oracle Extensions	Yes	No	Remove Entity Activity
RepeatUntil	BPEL Constructs	No	Yes	RepeatUntil Activity
Replay	Oracle Extensions	Yes	Yes	Replay Activity
Reply	BPEL Constructs	Yes	Yes	Reply Activity
Rethrow	BPEL Constructs	No	Yes	Rethrow Activity
Schedule Job	Oracle Extensions	Yes	Yes	Schedule Job
Scope	BPEL Constructs	Yes	Yes	Scope Activity
Sequence	BPEL Constructs	Yes	Yes	Sequence Activity
Signal	Oracle Extensions	Yes	Yes	Signal Activity
SMS	Oracle Extensions	Yes	Yes	SMS Activity
Switch	BPEL Constructs	Yes	No **Note**: Replaced by the if activity in BPEL 2.0.	Switch Activity
Terminate	BPEL Constructs	Yes	No **Note**: Replaced by the exit activity in BPEL 2.0	Terminate Activity
Throw	BPEL Constructs	Yes	Yes	Throw Activity
Translate	Oracle Extensions	Yes	Yes	Translate Activity
User Notification	Oracle Extensions	Yes	Yes	User Notification 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
Validate	Oracle Extensions (in BPEL 1.1) BPEL Constructs (in BPEL 2.0)	Yes	Yes	Validate Activity
Voice	Oracle Extensions	Yes	Yes	Voice Activity
Wait	BPEL Constructs	Yes	Yes	Wait Activity
While	BPEL Constructs	Yes	Yes	While Activity
XQuery Transform	Oracle Extensions	Yes	Yes	XQuery Transform Activity
XSLT Transform	Oracle Extensions	Yes	Yes	XSLT Transform Activity

For more information about activities, see the or the by visiting the following URL:

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

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.

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.

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.

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.

You can select when to execute a condition:

- Preassert: This condition is executed before the invoke or reply activity send out the outbound message.
- Postassert: This condition is executed after an invoke activity, receive activity, or onMessage branch receives the inbound message.

For more information, see the online help for this tab and Throwing Faults with Assertion Conditions.

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 Introduction to Correlation Sets in an Asynchronous Service.

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. The following example provides details.

```
<invoke>
. . .
   <documentation>
      Invokes the credit rating service partner link
   </documentation>
. . .
```

Note:

This tab is only available in BPEL 2.0 projects.

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 adapters, such as Advanced Queuing (AQ), file, FTP, MQ, and Java Message Service (JMS).

For more information, see the online help for this tab and *Understanding Technology Adapters*

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 components such as Oracle BPEL Process Manager, Oracle Mediator, Oracle JCA adapters, REST adapters, and Oracle B2B.

For more information, see the online help for this tab and Propagating Normalized Message Properties Through Message Headers.

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.

For more information, see the online help for this tab and Specifying XPath Expressions to Bypass Activity Execution.

Sources 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 has completed.

For more information, see the online help for this tab and Synchronizing the Execution of Activities in a Flow Activity.

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.

For more information, see the online help for this tab and Setting Timeouts for Request-Reply and In-Only Operations in Receive Activities.

Using the Native Format Builder Wizard Outside of Adapter Configuration

The Native Format Builder wizard enables you to create a native XSD schema file. You can now invoke the Native Format Builder wizard outside of adapter creation to create new schemas and edit existing schemas.

To create a native format schema from the Applications Window:

1. From the Oracle JDeveloper main menu, select **File** > **New**.
2. From the **Categories** list, select **SOA Tier** > **Interfaces**.
3. Click **NXSD Schema** to invoke the Native Format Builder wizard.
4. On the Welcome page, click **Next**.
5. On the File Name and Directory page, specify the file name of the schema (for example, `addresses_schema.xsd`) and directory path.
6. Follow the remaining pages of the wizard to create the native format schema.

For more information about the Native Format Builder wizard, see Chapter "Native Format Builder Wizard" of *Understanding Technology Adapters*.

To edit an existing native format schema from the Applications Window:

You can access the Native Format Builder wizard for schema editing from the Applications window. The context menu option **Edit NXSD** is available for selection if the schema file is detected to be a native format schema file.

1. In the Applications window, right-click a native format schema file (for example, **addresses.xsd**).
2. Select **Edit NXSD**.

For more information about the Native Format Builder wizard, see Chapter "Native Format Builder Wizard" of *Understanding Technology Adapters*.

Assign Activity

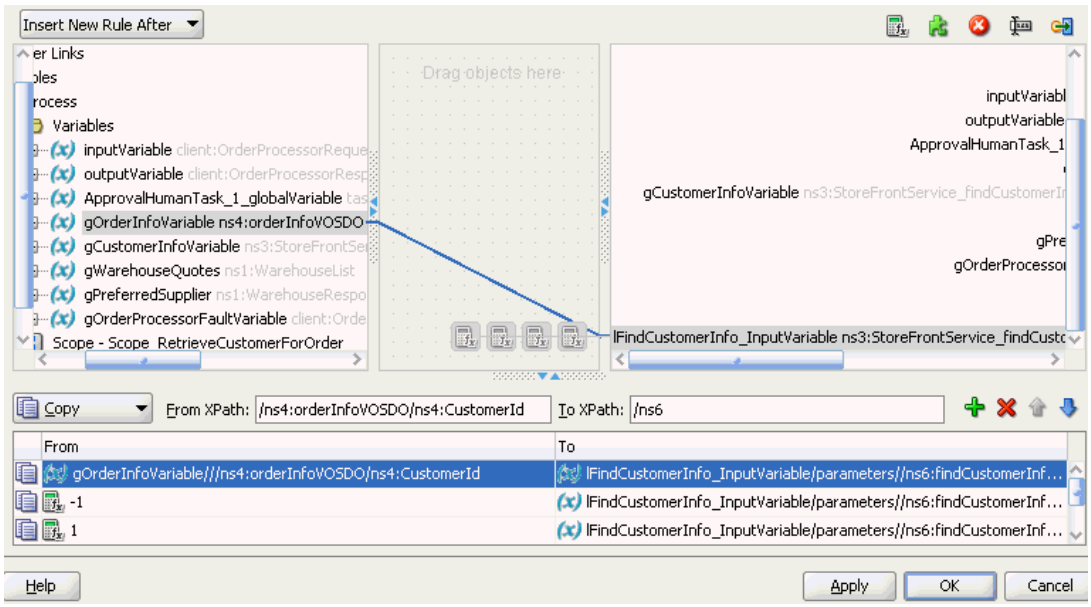
This activity provides a method for data manipulation, such as copying the contents of one variable to another. Copy operations enable you to transfer information between variables, expressions, endpoints, and other elements.

Figure A-2 shows the **Copy Rules** tab of the Assign dialog for BPEL 1.1. You create a mapping between source and target nodes in the tree in either of the following ways:

- 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.
- Select the source node, select the target node, and then click the **Add** icon above the table at the bottom of the dialog. The mapping is then added to the table and the connecting line between the nodes is drawn in the tree.

The copy rule is displayed in the **From** and **To** sections at the bottom of the dialog.

Figure A-2 Copy Rules Tab of Edit Assign 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. By default, the center canvas is open. If it is closed, drag the bars open to display the center canvas.

- **Expression** icon: Drag this icon to a target node to invoke the Expression Builder dialog for assigning an XPath expression to that node. You can also drag this icon to the center canvas to invoke this dialog, specify the expression, save and close the dialog, and then drag the icon to the target 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. You can also drag

this icon to the center canvas to invoke this dialog, specify the value, save and close the dialog, and then drag the icon to the target node.

- **Remove** icon: Drag this icon to a target node to create a `bpelx:remove` extension rule. You can also drag this icon to the center canvas to invoke this dialog, specify the rule, save and close the dialog, and then drag the icon to the target node.
- **Rename** icon: Drag this icon to rename a target node. This adds a `bpelx:rename` extension rule with an `elementTo` attribute. You can also drag this icon to the center canvas to invoke a dialog, specify the rule, save and close the dialog, and then drag the icon to the target node.
- **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 drag this icon to the center canvas to invoke a dialog, specify the rule, save and close the dialog, and then drag the icon to the target node.

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

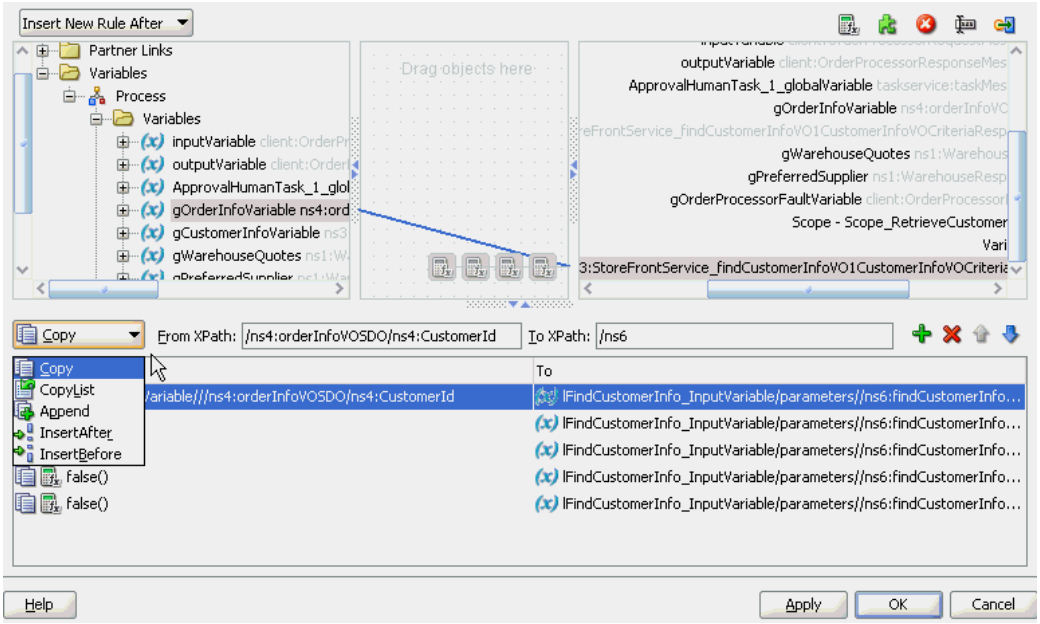
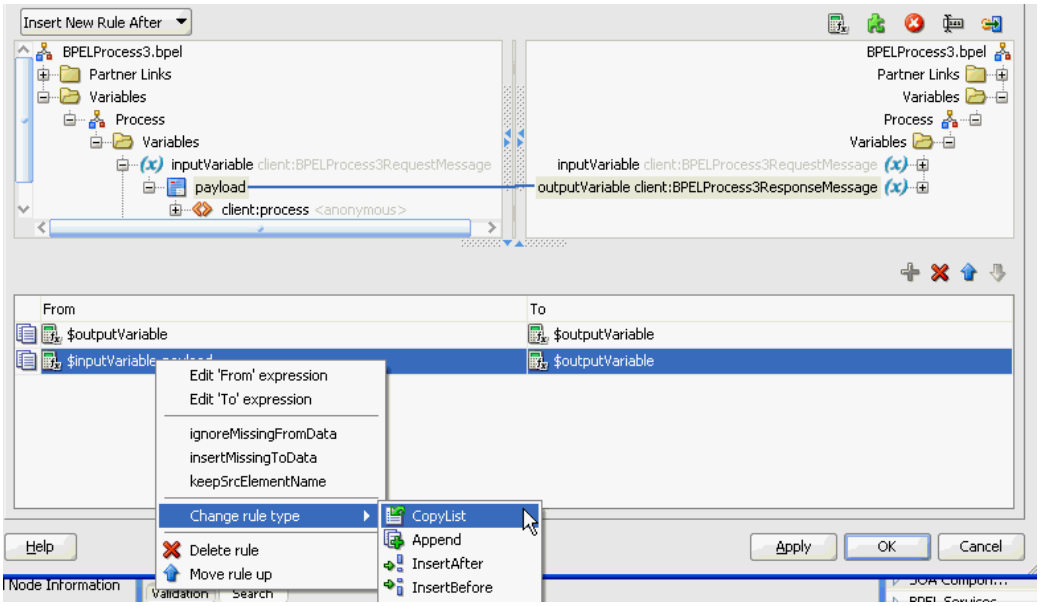


Figure A-4 shows how you select an extension type in BPEL 2.0. You right-click a copy rule, select **Change rule type**, and then select the appropriate extension.

Figure A-4 Copy Rule Converted to bpelx Extension in BPEL 2.0



For more information about manipulating XML data with `bpelx` extensions, see 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. For more information, see ignoreMissingFromData Attribute.
- **insertMissingToData**: Select this option to toggle the `insertMissingToData` attribute on the copy rule on and off. For more information, see Section insertMissingToData Attribute.
- **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 How to Use Assign Extension Attributes.

The icons above the **To** section enable you to add, 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 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.

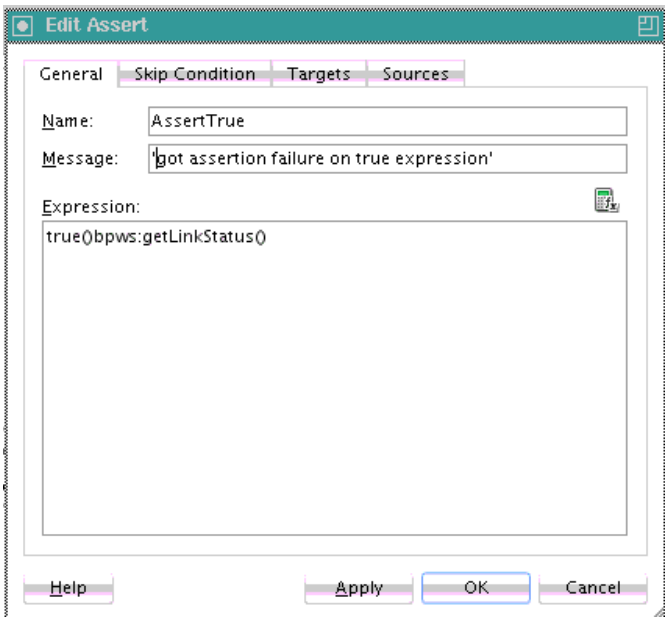
Assert Activity

This activity enables you to perform an assertion on a specified expression.

This is a standalone activity in which to specify assertions. This activity can be placed anywhere in the BPEL process flow. You can also specify assertions in message exchange activities from the **Assertions** tab in invoke activities, reply activities, receive activities, and the onMessage branch of pick and scope activities.

Figure A-5 shows the Assert dialog.

Figure A-5 Assert Dialog



For more information about the standalone assert activity, see Assertion Conditions in a Standalone Assert Activity and What Happens When You Create Assertion Conditions.

Bind Entity Activity

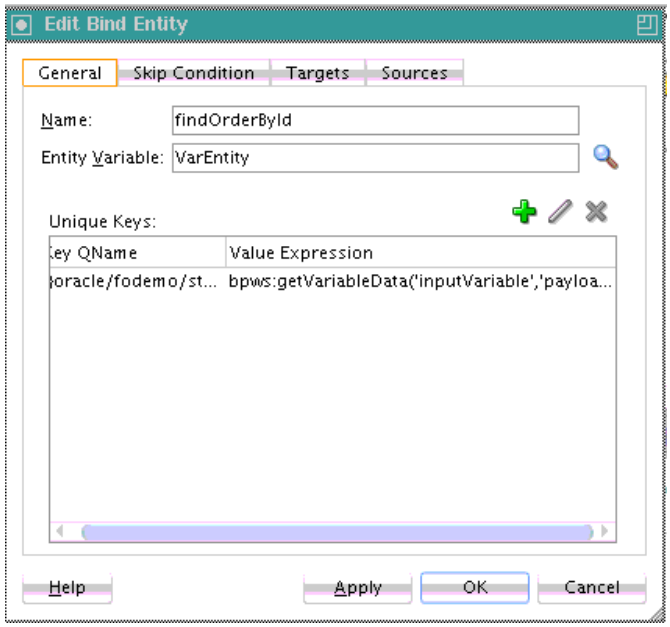
This activity enables you to select the entity variable to act as the data handle to access and plug in different data provider service technologies.

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

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

Figure A-6 shows the Bind Entity dialog.

Figure A-6 Bind Entity Dialog

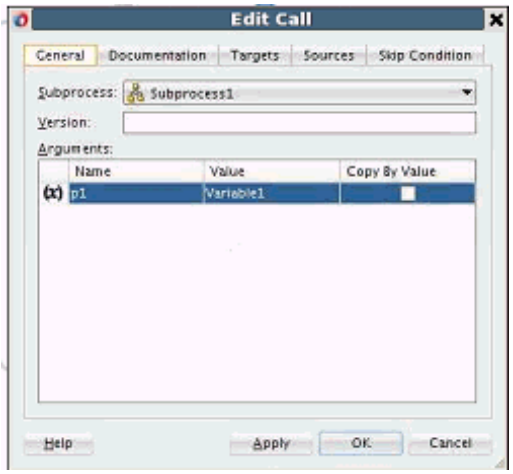


Call Activity

This activity enables you to execute referenced subprocess code in standalone and inline subprocesses in BPEL 2.0. A subprocess is a fragment of BPEL code that can be reused within a particular processor by separate processes.

Figure A-7 shows the Edit Call dialog.

Figure A-7 Edit Call Dialog



For more information about the call activity, see Introduction to Standalone and Inline BPEL Subprocess Invocations and Creating Standalone and Inline BPEL Subprocesses in a BPEL Process.

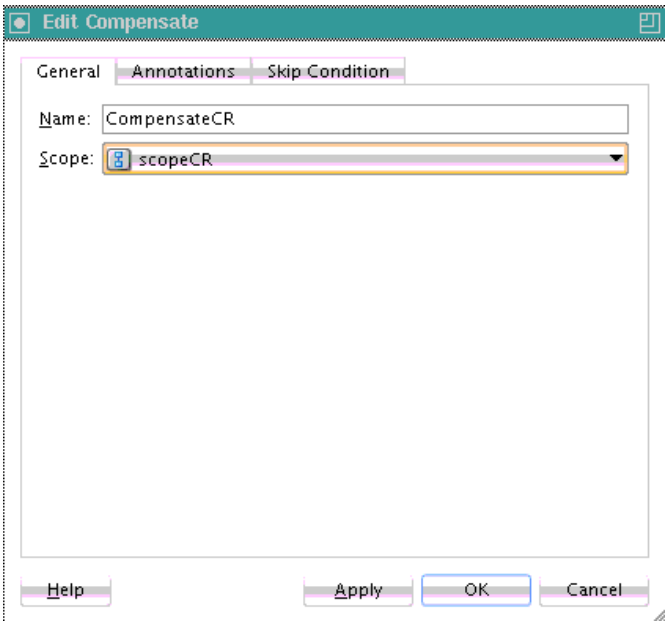
Compensate Activity

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-8 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 to invoke the compensation handler.

Figure A-8 Compensate Dialog



In BPEL 2.0, the Compensate dialog includes a **Documentation** tab.

For more information about the compensate activity, see Using Compensation After Undoing a Series of Operations.

CompensateScope Activity

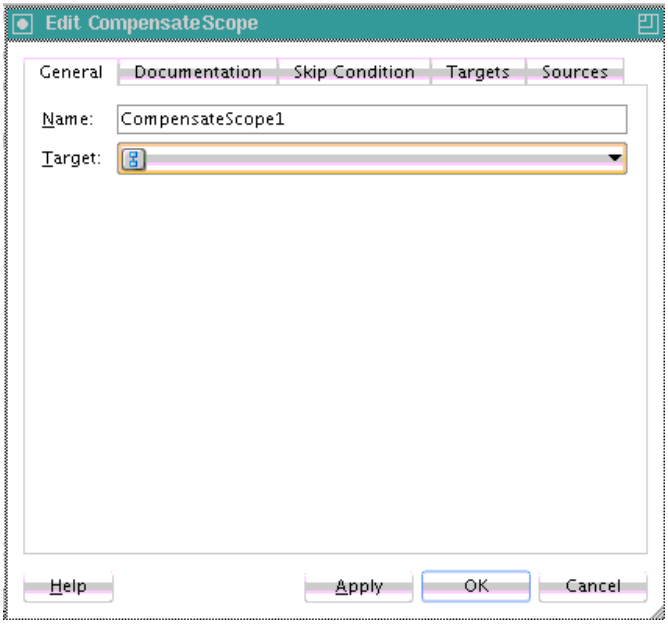
This activity enables you to start compensation on a specified inner scope that has already completed successfully. Only use this activity from within a fault handler, another compensation handler, or a termination handler.

Note:

This activity is only supported in BPEL 2.0 projects.

Figure A-9 shows the CompensateScope dialog.

Figure A-9 CompensateScope Dialog



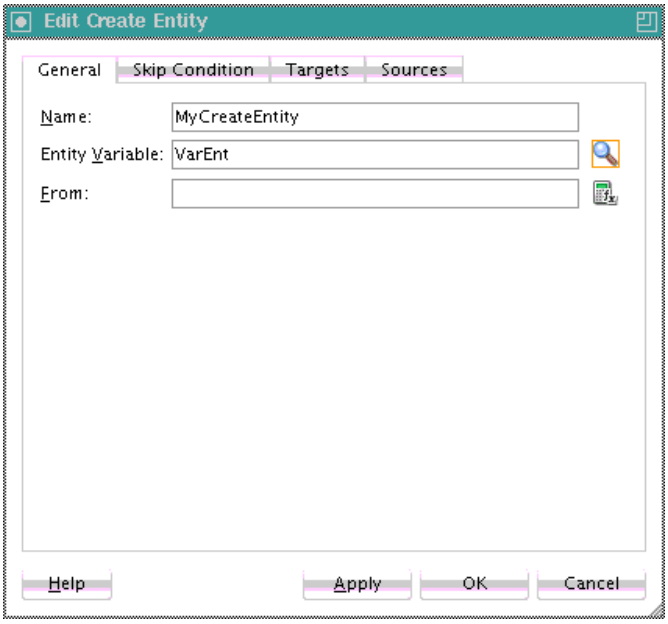
For more information about the compensateScope activity, see Using Compensation After Undoing a Series of Operations.

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.

Figure A-10 shows the Create Entity dialog.

Figure A-10 Create Entity Dialog



For more information, see Delegating XML Data Operations to Data Provider Services.

Dehydrate Activity

By default, dehydration points are set on activities such as a receive, onMessage, onAlarm, and wait. 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:

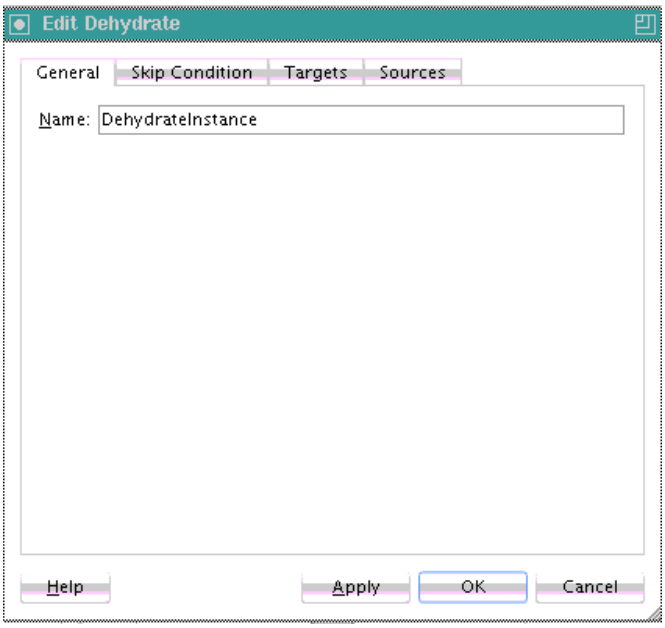
```
<bpelx:dehydrate name="DehydrateInstance"/>
```

For BPEL projects that support BPEL version 2.0, the syntax is as shown in the following example:

```
<extensionActivity>
      <bpelx:dehydrate name="DehydrateInstance"/>
 </extensionActivity>
```

Figure A-11 shows the Dehydrate dialog in BPEL 2.0.

Figure A-11 Dehydrate Dialog

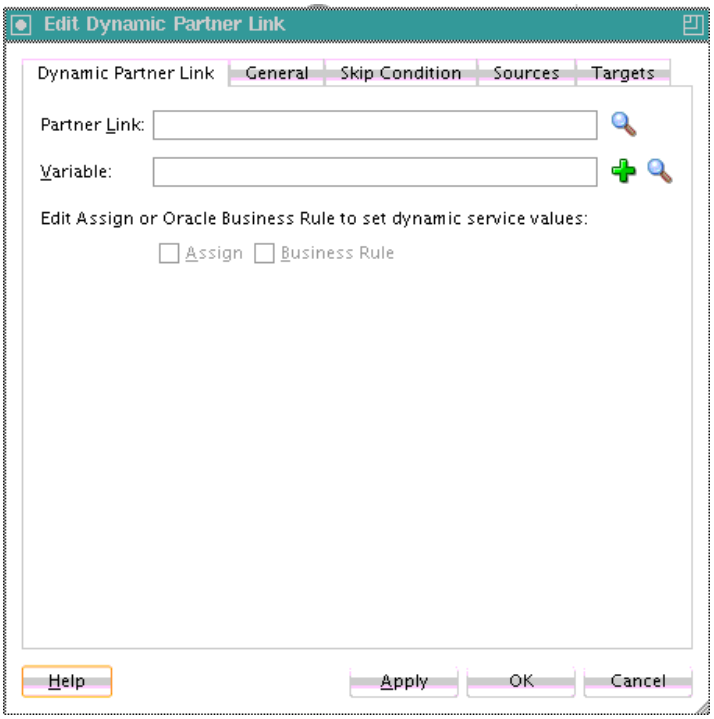


Dynamic Partner Link Activity

This activity enables you to dynamically assign an endpoint reference to a partner link for use at runtime in BPEL version 1.1.

Figure A-12 shows the Dynamic Partner Link dialog in BPEL 1.1.

Figure A-12 Dynamic Partner Link Dialog



For more information, see Creating a Dynamic Partner Link at Design Time for Use at Runtime.

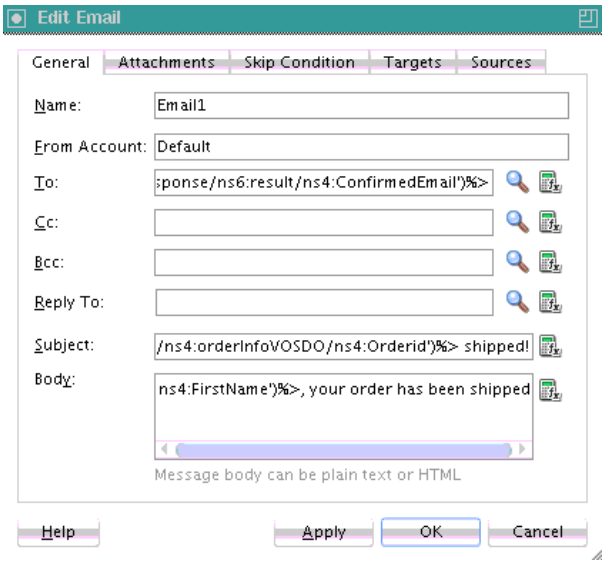
Email Activity

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-13 shows the Email dialog in BPEL 2.0.

Figure A-13 Email Dialog



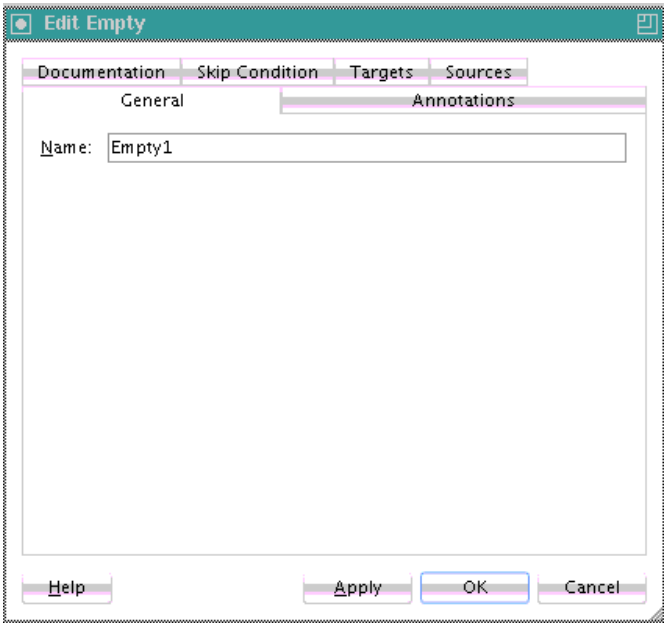
For more information about the email activity, see How To Configure the Email Notification Channel.

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-14 shows the Empty dialog in BPEL 2.0.

Figure A-14 Empty Dialog



For more information about the empty activity, see How to Insert No-Op Instructions into a Business Process with an Empty Activity.

Exit Activity

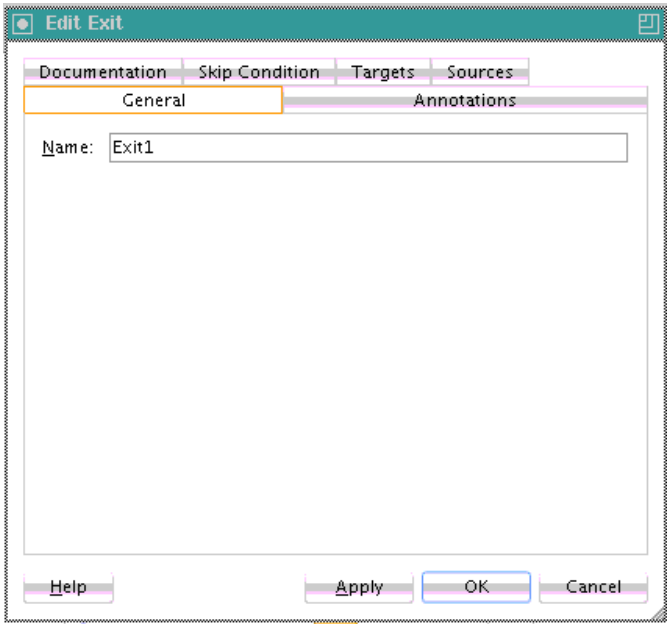
This activity enables you to immediately end all currently running activities on all parallel branches without involving any termination handling, fault handling, or compensation handling mechanisms.

Note:

This activity replaces the terminate activity in BPEL 2.0 projects.

Figure A-15 shows the Exit dialog.

Figure A-15 Exit Dialog



For more information about the exit activity, see Immediately Ending a Business Process Instance with the Exit Activity in BPEL 2.0.

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-16 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 as shown in Figure A-17.

Figure A-16 Flow Dialog (At Time of Creation)

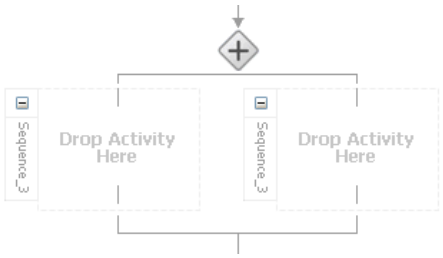
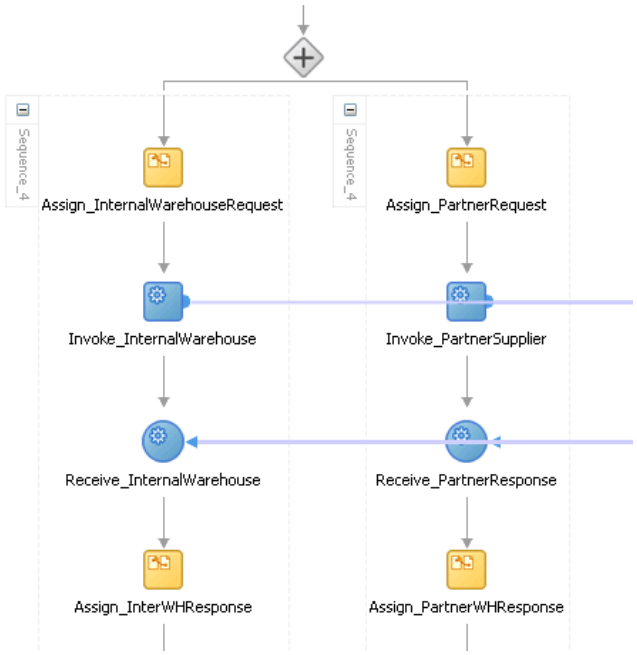


Figure A-17 Flow Dialog (After Design Completion)



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 Creating a Parallel Flow.

FlowN Activity

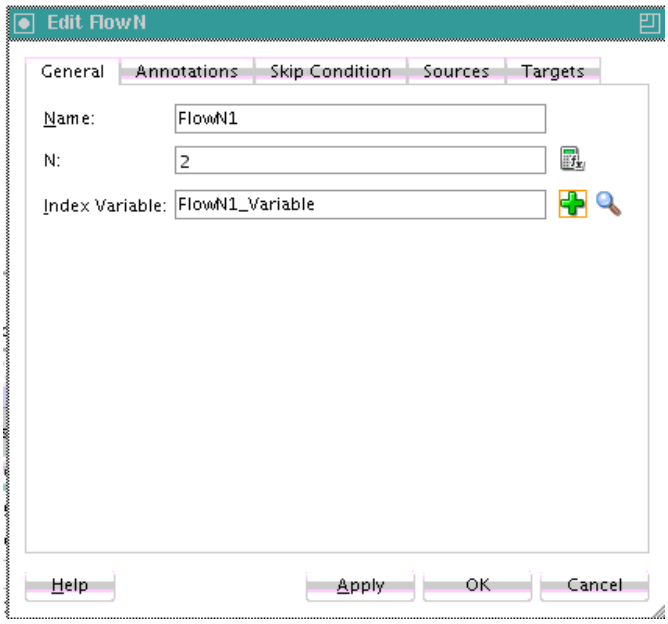
This activity enables you to create multiple flows equal to the value of `N`, which is defined at runtime based on the data available and logic within the process. An index variable increments each time a new branch is created, until the index variable reaches the value of `N`.

Note:

This activity is replaced by the forEach activity in BPEL 2.0 projects.

Figure A-18 shows the FlowN dialog.

Figure A-18 FlowN Dialog



For more information about the flowN activity, see Customizing the Number of Flow Activities with the flowN Activity in BPEL 1.1.

forEach Activity

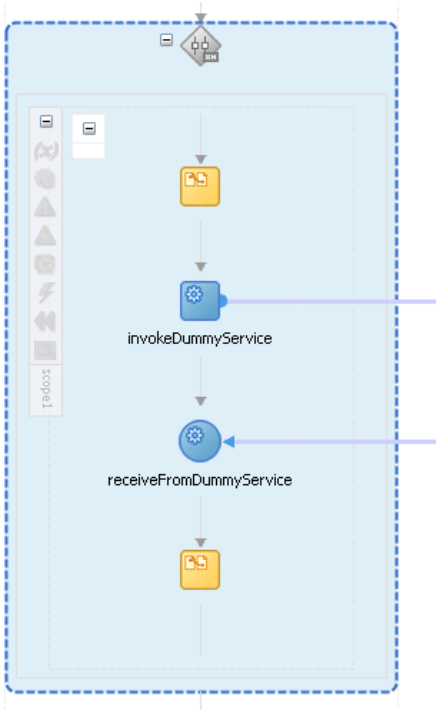
This activity enables you to process multiple sets of activities sequentially or in parallel. The forEach activity executes its contained (child) scope activity exactly *N*+1 times, where *N* equals the final counter value minus the starting counter value that you specify in the **Counter Values** tab 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-19 shows a forEach activity with its contained scope.

Figure A-19 forEach Activity



For more information about the forEach activity, see Processing Multiple Sets of Activities with the forEach Activity in BPEL 2.0.

If Activity

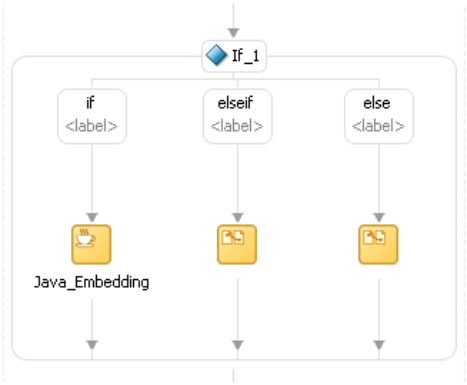
This activity enables you to define conditional behavior for specific activities to decide between two or more branches. Only one activity is selected for execution from a set of branches.

Note:

This activity replaces the switch activity in BPEL 2.0 projects.

Figure A-20 shows an if activity with the following defined if, elseif, and else branches.

Figure A-20 If Activity

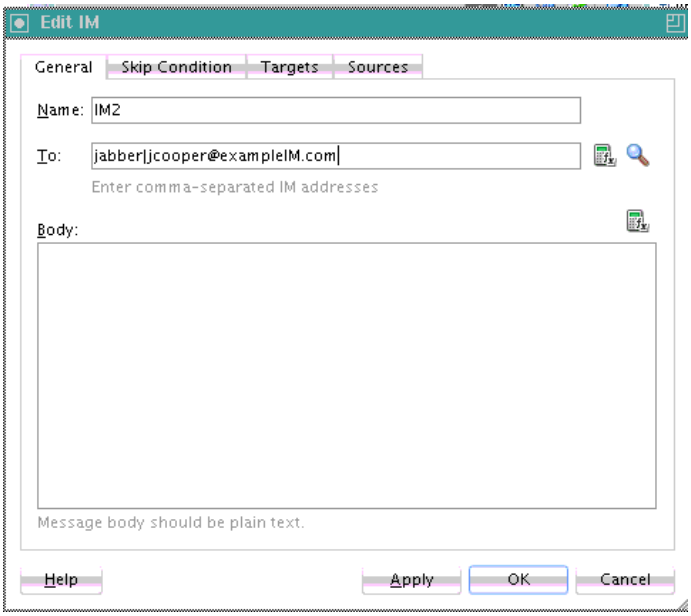


For more information about the if activity, see Defining Conditional Branching with the If Activity in BPEL 2.0.

IM Activity

This activity enables you to send an automatic, asynchronous instant message (IM) notification to a user, group, or destination address. Figure A-21 shows the IM dialog.

Figure A-21 IM Dialog



For more information, see How to Configure the IM Notification Channel.

Invoke Activity

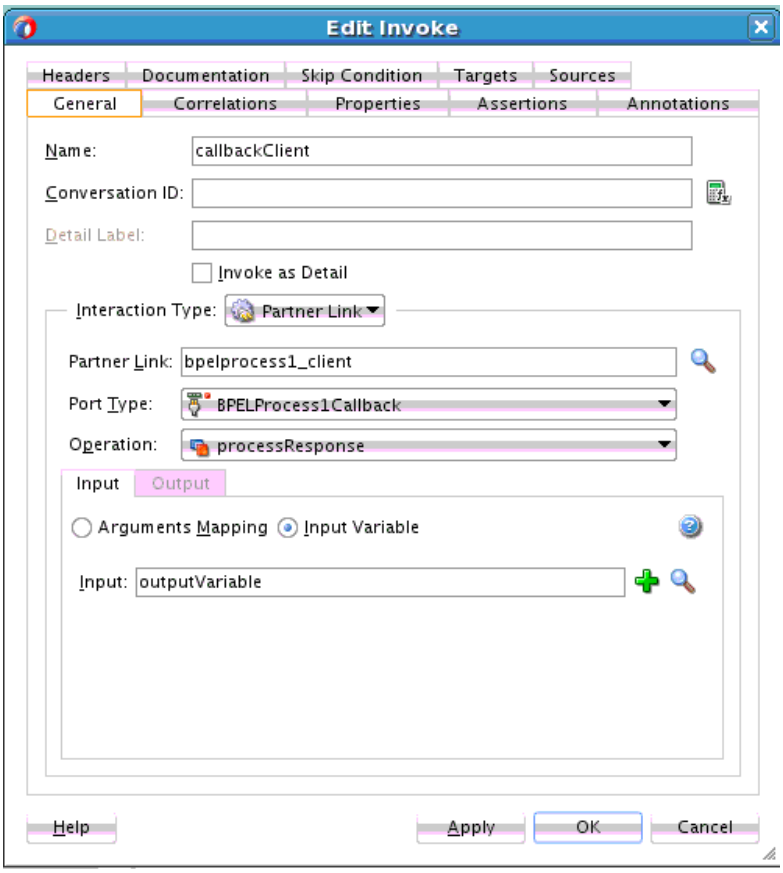
This activity enables you to specify an operation you want to invoke for the service (identified by its partner link). The operation can be one-way or request-response on a port provided by the service. You can also automatically create variables in an invoke activity. An invoke activity invokes a synchronous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

Figure A-22 shows the Invoke dialog in BPEL 2.0. 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 perform.
- Automatically create a variable or select an existing variable in which to transport the data (payload).

Figure A-22 Invoke Dialog



For more information about the invoke activity, see the following:

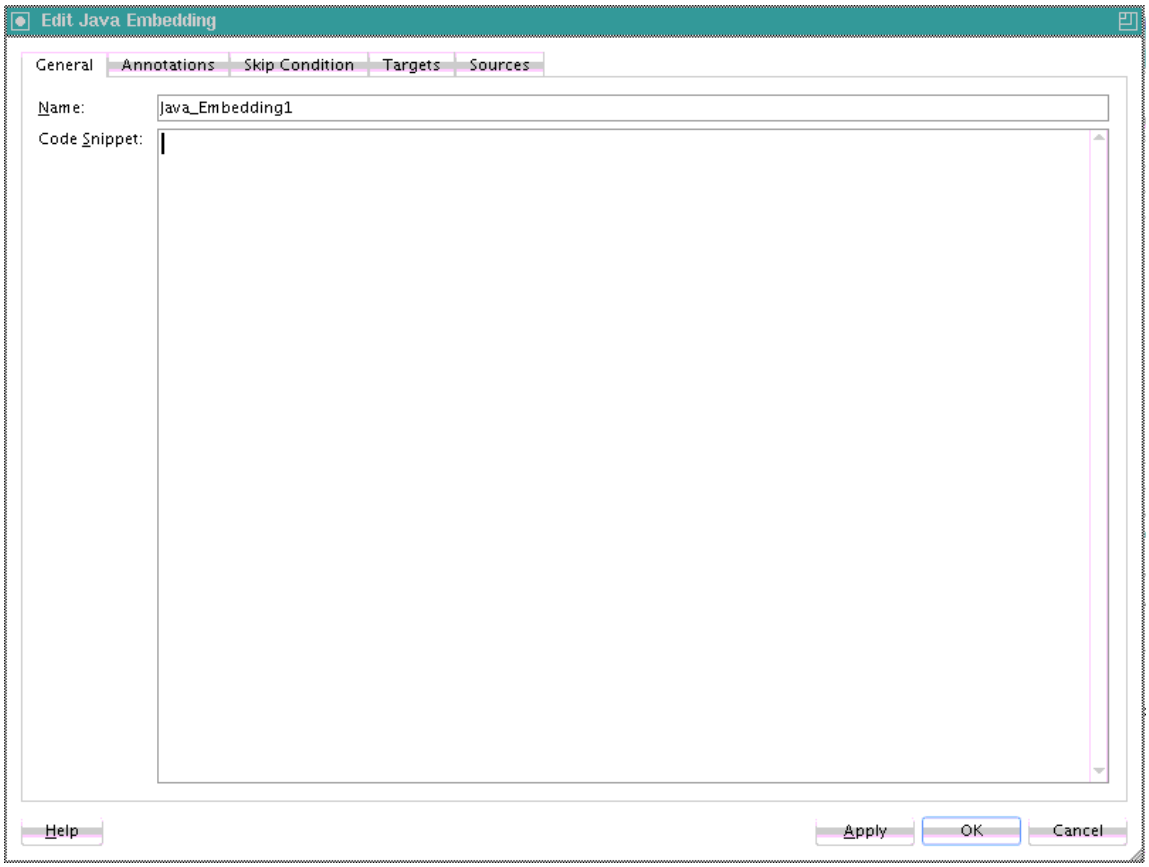
- Introduction to Interaction Patterns in a BPEL Process
- Mapping WSDL Message Parts in BPEL 2.0
- Invoke Activity for Performing a Request
- Adding an Invoke Activity
- How to Return a Fault in an Asynchronous Interaction
- Throwing Faults with Assertion Conditions

Java Embedding Activity

This activity enables you to add custom Java code to a BPEL process using the Java BPEL 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-23 shows the Edit Java Embedding dialog in BPEL 2.0.

Figure A-23 Edit Java Embedding Dialog



For more information about the Java embedding activity, see Incorporating Java and Java EE Code in a BPEL Process.

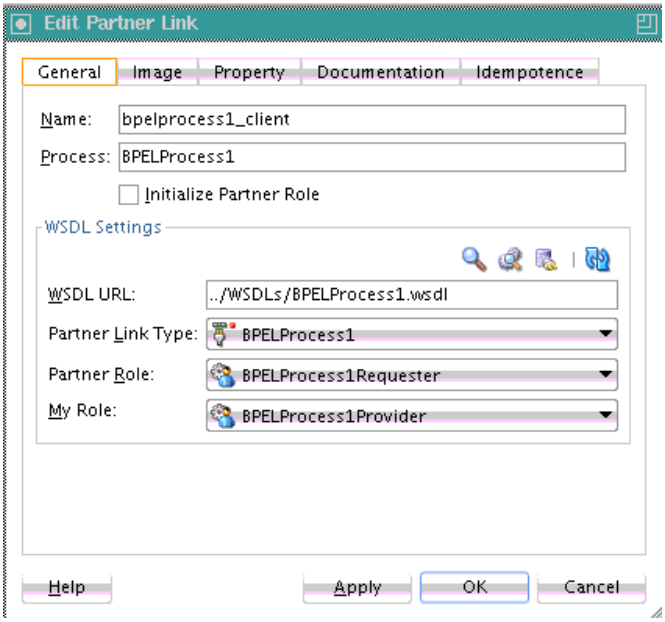
Partner Link Activity

This activity 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 create 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-24 shows the Partner Link dialog in BPEL 2.0. 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-24 Partner Link Activity



For more information about partner links, see Invoking an Asynchronous Web Service from a BPEL Process.

Phase Activity

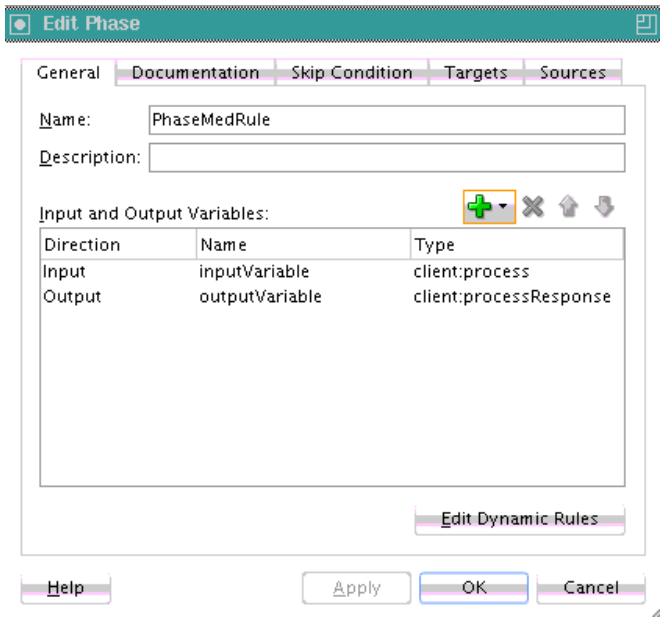
This activity creates Oracle Mediator and business rules service components for integration with a BPEL process. You create message request input and message response output variables and design business rules for evaluating variable content for the BPEL process.

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 that is automatically designed to invoke an Oracle Mediator partner link in the BPEL process.
- An Oracle Mediator partner link that 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 . Oracle Mediator also displays as a service component in the .
- A business rules service component that 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-25 shows the Phase dialog in BPEL 2.0.

Figure A-25 Phase Dialog



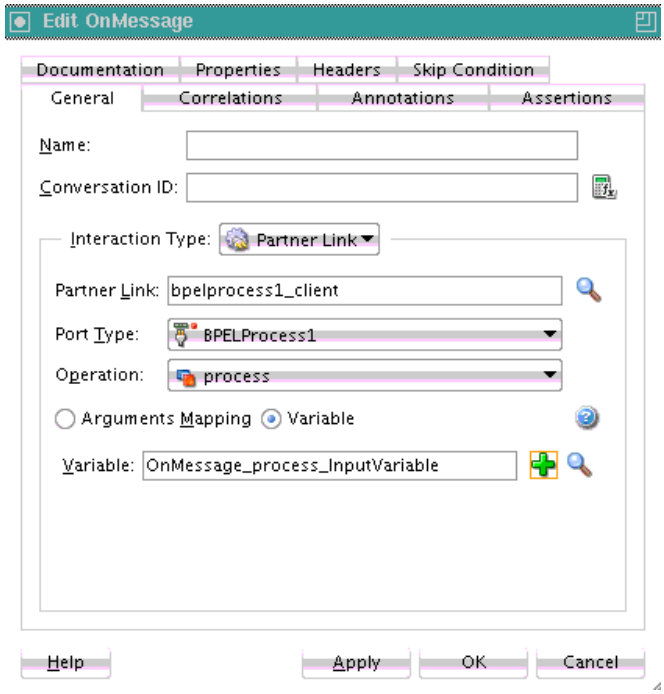
For more information, see Creating Dynamic Business Processes .

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 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 2.0, the dialog shown in Figure A-26 appears.

Figure A-26 OnMessage Dialog



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.

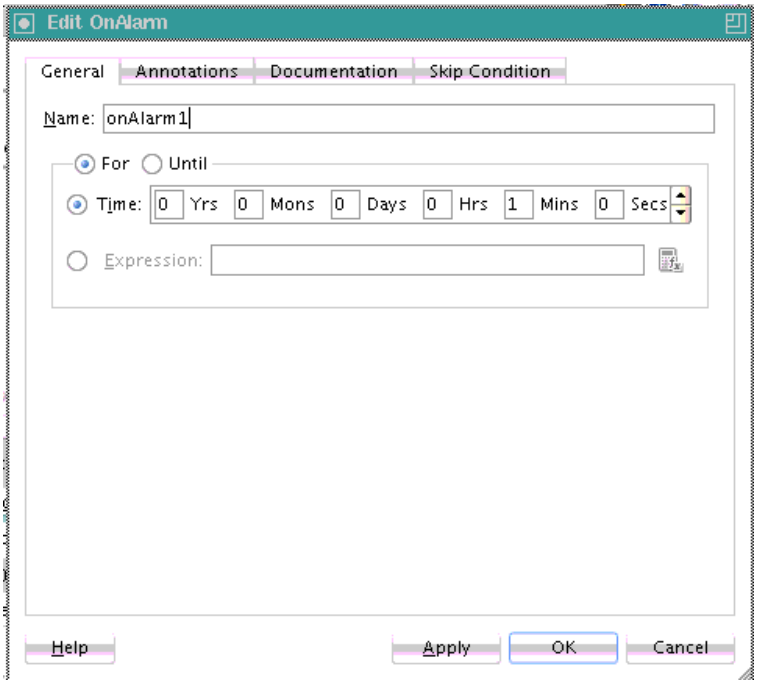
- OnAlarm (Does not automatically display; you must manually add this branch by selecting the **Pick** activity icon and clicking the **Add OnAlarm** icon.)

  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-27 shows the OnAlarm dialog of the pick activity in BPEL 2.0.

Figure A-27 OnAlarm Branch Dialog of 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.

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.

To put the correlation syntax before the assign activity:

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:

- Introduction to Interaction Patterns in a BPEL Process
- Mapping WSDL Message Parts in BPEL 2.0
- Throwing Faults with Assertion Conditions
- Selecting Between Continuing or Waiting on a Process with a Pick Activity
- Setting Timeouts for Durable Synchronous Processes

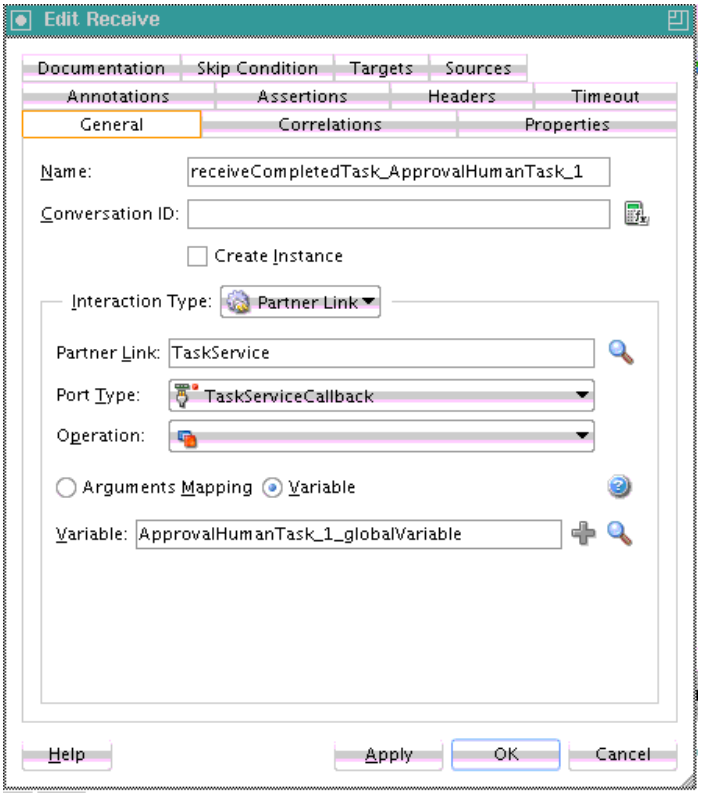
Receive Activity

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 approval 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.

Figure A-28 shows the Receive dialog in BPEL 2.0. 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-28 Receive Dialog



For more information about the receive activity, see the following:

- Introduction to Interaction Patterns in a BPEL Process
- Mapping WSDL Message Parts in BPEL 2.0
- Adding a Receive Activity
- Throwing Faults with Assertion Conditions

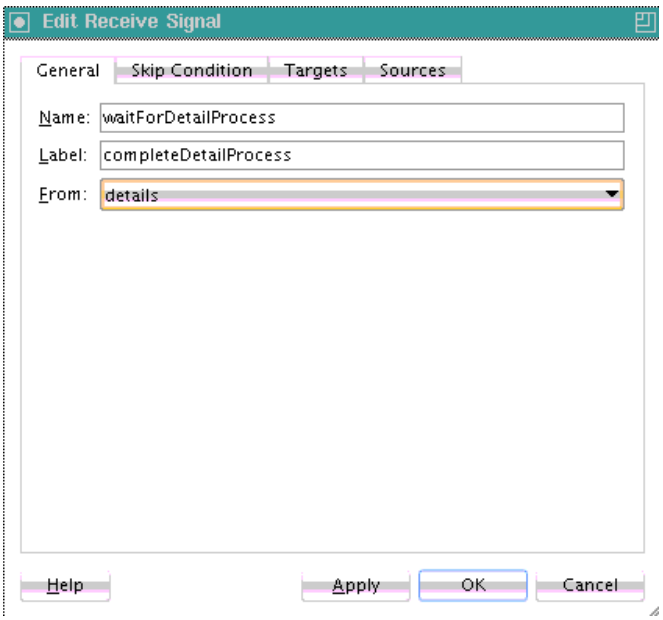
- Setting Timeouts for Request-Reply and In-Only Operations in Receive Activities

Receive Signal Activity

Use this activity in detail processes to wait for the notification signal from the master process to begin processing and in a master process to wait for the notification signal from all detail processes indicating that processing has completed.

Figure A-29 shows the Edit Receive Signal dialog.

Figure A-29 Receive Signal Dialog



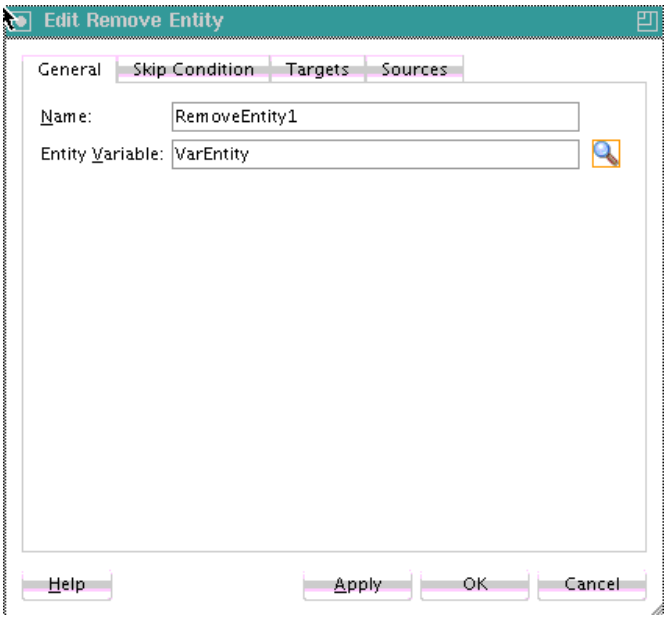
For more information, see Coordinating Master and Detail Processes .

Remove Entity Activity

This activity enables you to remove an entity variable. This action removes the row.

Figure A-30 shows the Remove Entity dialog.

Figure A-30 Remove Entity Dialog



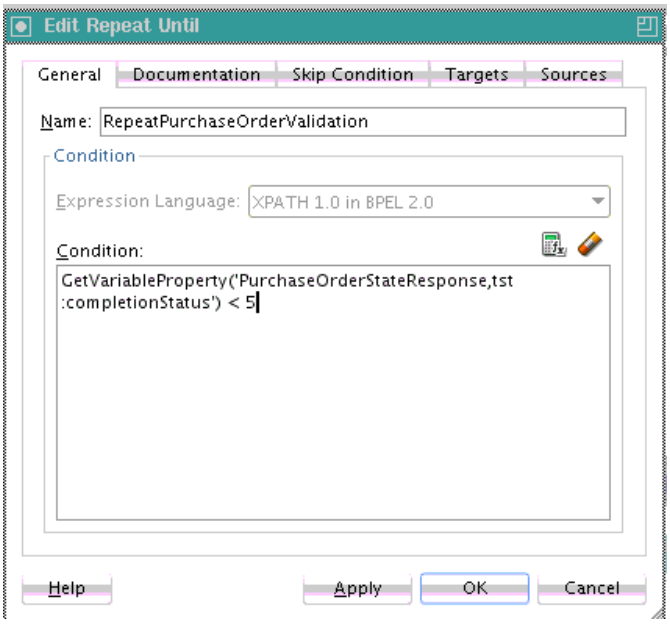
RepeatUntil Activity

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-31 shows the Repeat Until dialog.

Note:

This activity is only supported in BPEL 2.0 projects.

Figure A-31 Repeat Until Dialog



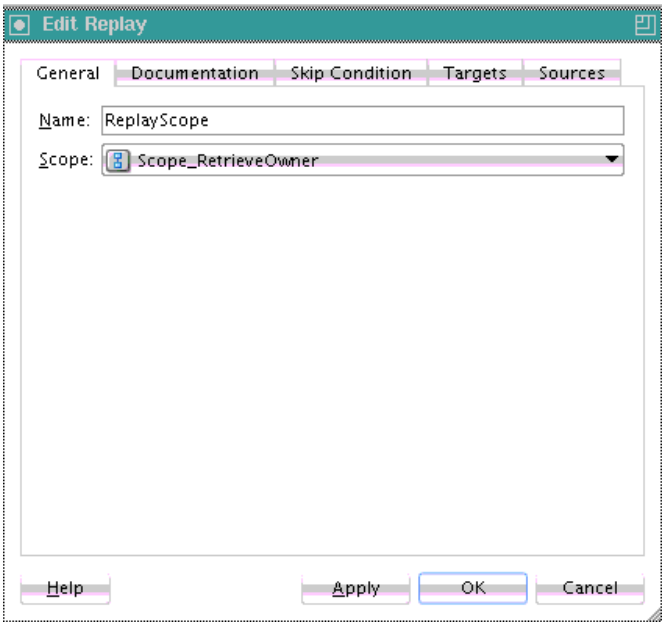
For more information about the repeatUntil activity, see, Defining Conditional Branching with the repeatUntil Activity.

Replay Activity

This activity enables you to re-execute the activities inside a selected scope.

Figure A-32 shows the Replay dialog in BPEL 2.0.

Figure A-32 Replay Dialog



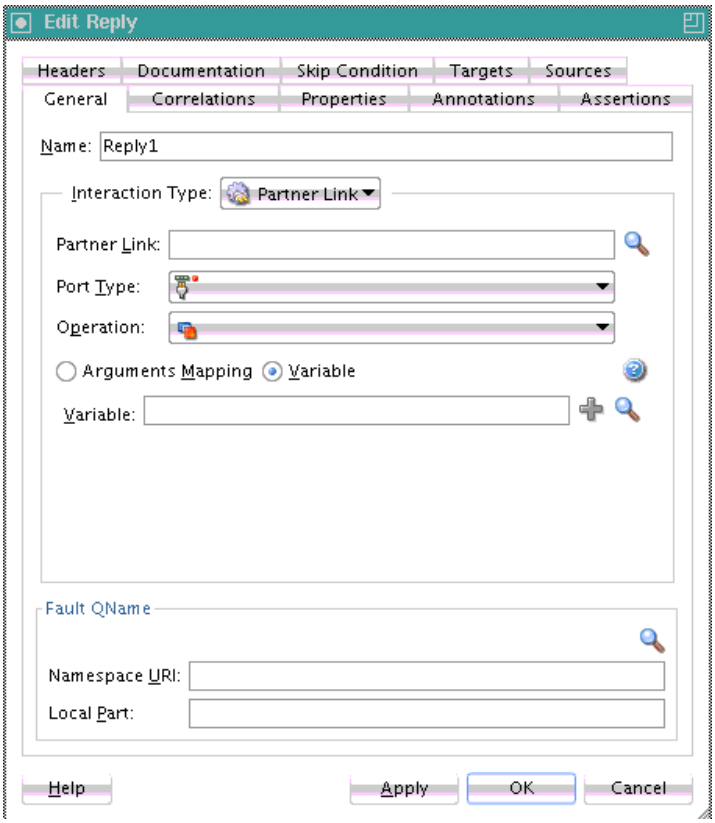
For more information about the replay activity, see Re-executing Activities in a Scope Activity with the Replay Activity.

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-33 shows the Reply dialog in BPEL 2.0.

Figure A-33 Reply Dialog



For more information about the reply activity, see the following:

- Introduction to Interaction Patterns in a BPEL Process
- Mapping WSDL Message Parts in BPEL 2.0
- How to Return a Fault in a Synchronous Interaction

Rethrow Activity

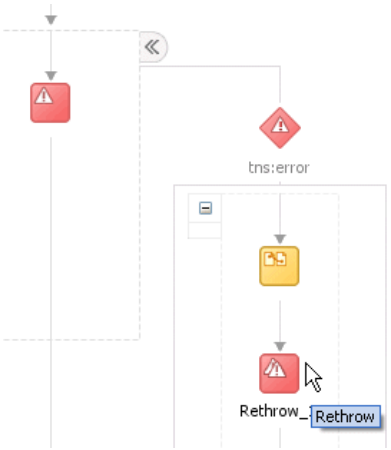
This activity enables you to rethrow a fault originally captured by the immediately enclosing fault handler.

Note:

This activity is only supported in BPEL 2.0 projects.

Figure A-34 shows a rethrow activity within a fault handler (catch activity).

Figure A-34 Rethrow Activity

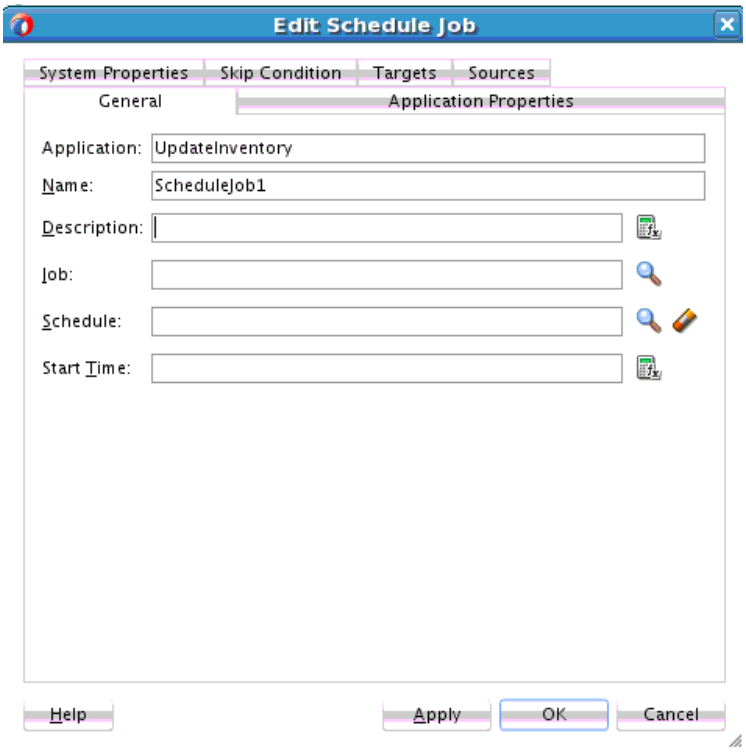


For more information about rethrowing faults, see Rethrowing Faults with the Rethrow Activity.

Schedule Job

This activity enables you to schedule an Oracle Enterprise Scheduler job in a BPEL process. Figure A-35 shows the Schedule Job dialog.

Figure A-35 Schedule Job Dialog



For more information, see Invoking an Oracle Enterprise Scheduler Job in a BPEL Process.

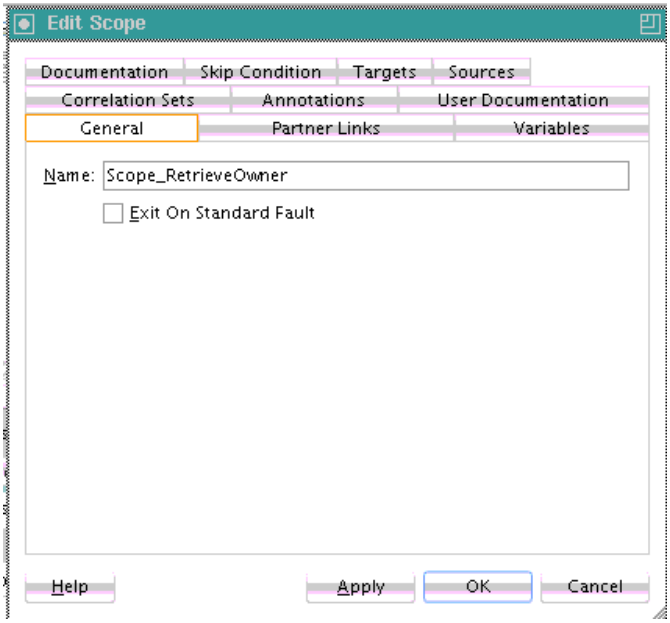
Scope Activity

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 of arbitrary depth. The scope is shared by all the nested activities.

Figure A-36 shows the Scope dialog in BPEL 2.0. Define appropriate activities inside the scope activity.

Figure A-36 Scope Dialog



Fault handling is associated with a scope activity. The goal is to undo the incomplete and unsuccessful work of a scope activity in which a fault has occurred. You define catch activities in a scope activity to create a set of custom fault-handling activities. Each catch activity is defined to intercept a specific type of fault.

Figure A-37 shows the **Add Catch** icon inside a scope activity. Figure A-38 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-37 Creating a Catch Branch

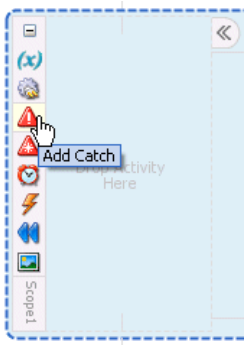
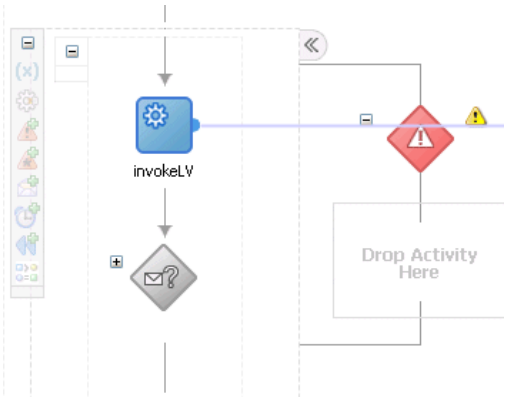


Figure A-38 Catch Activity Icon



For more information about the scope activity and fault handling, see the following:

- Introduction to Interaction Patterns in a BPEL Process
- Mapping WSDL Message Parts in BPEL 2.0
- Managing a Group of Activities with a Scope Activity

Sequence Activity

This activity enables you to define a collection of activities to perform 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-39 appears. Drag and define appropriate activities inside the sequence activity.

Figure A-39 Sequence Activity

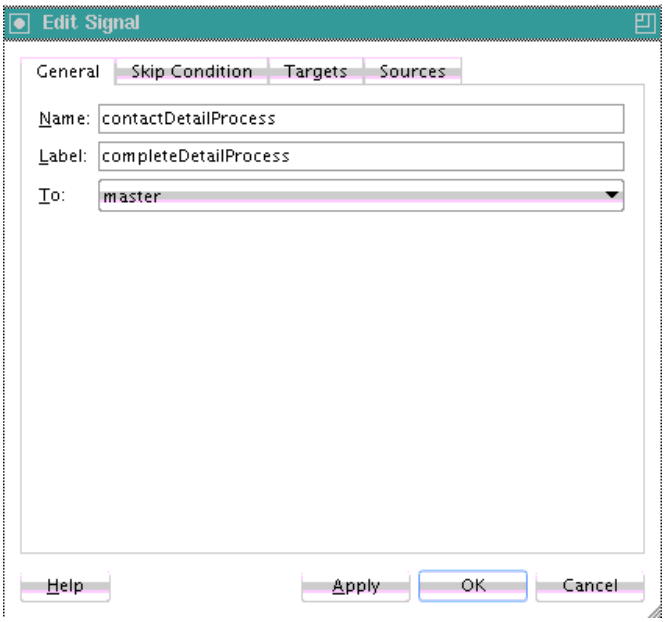
For more information about the sequence activity, see the following:

- Introduction to Interaction Patterns in a BPEL Process
- Creating a Parallel Flow

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-40 shows the Edit Signal dialog.

Figure A-40 Signal Dialog



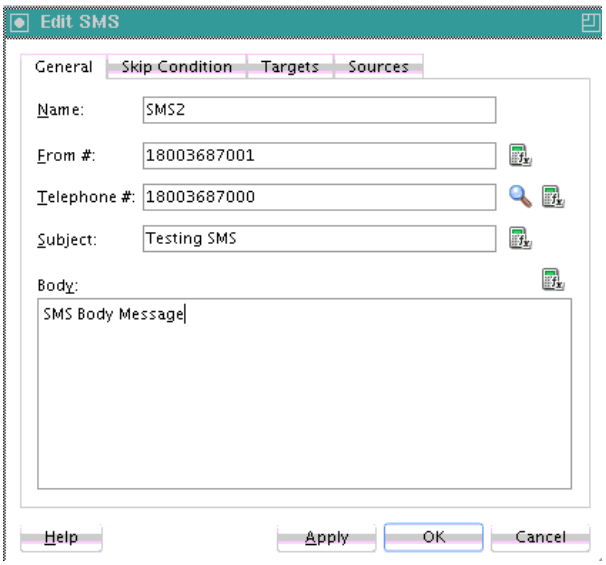
For more information, see Coordinating Master and Detail Processes .

SMS Activity

This activity enables you to send a short message system (SMS) notification about an event.

Figure A-41 shows the SMS dialog in BPEL 2.0.

Figure A-41 SMS Dialog



For more information about the SMS activity, see How to Configure the SMS Notification Channel.

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.

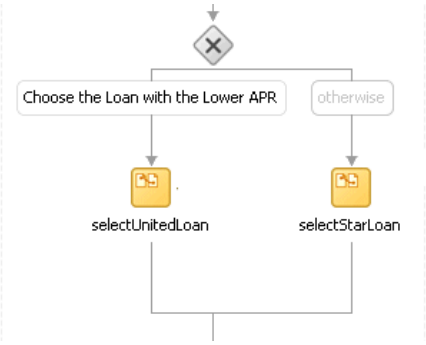
A switch activity differs in functionality from a flow activity. For example, a flow activity enables a process to gather two loan offers at the same time, but does not compare their values. To compare and make decisions on the values of the two offers, a switch activity is used. The first branch is 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-42 shows a switch activity with the following defined branches.

Figure A-42 Switch Activity



For more information about the switch activity, see the following:

- Introduction to Interaction Patterns in a BPEL Process
- Defining Conditional Branching with the Switch Activity in BPEL 1.1

Terminate Activity

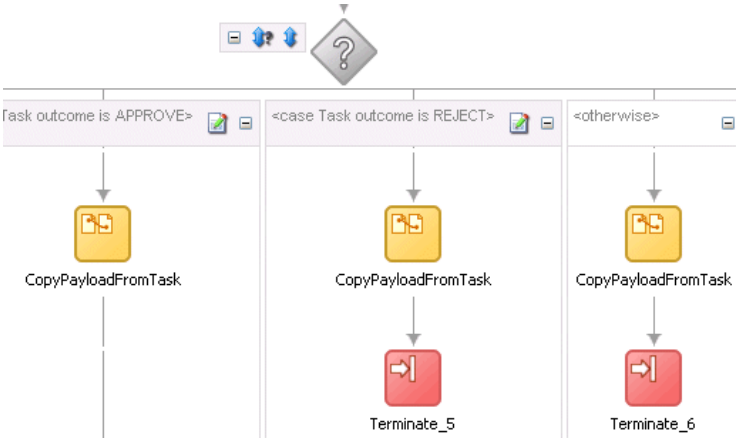
This activity enables you to end the tasks of an activity (for example, the fault handling tasks in a catch branch). For example, if a client's bad credit history is identified or a social security number is identified as invalid, a loan application process is terminated, and the client's loan application document is never submitted to the service loan providers.

Note:

- The terminate activity is replaced by the exit activity in BPEL 2.0 projects.
- Do not use the terminate activity with a synchronous BPEL process because it can lead to timeouts.

Figure A-43 shows several terminate activities in the otherwise branch of a switch activity.

Figure A-43 Terminate Activity



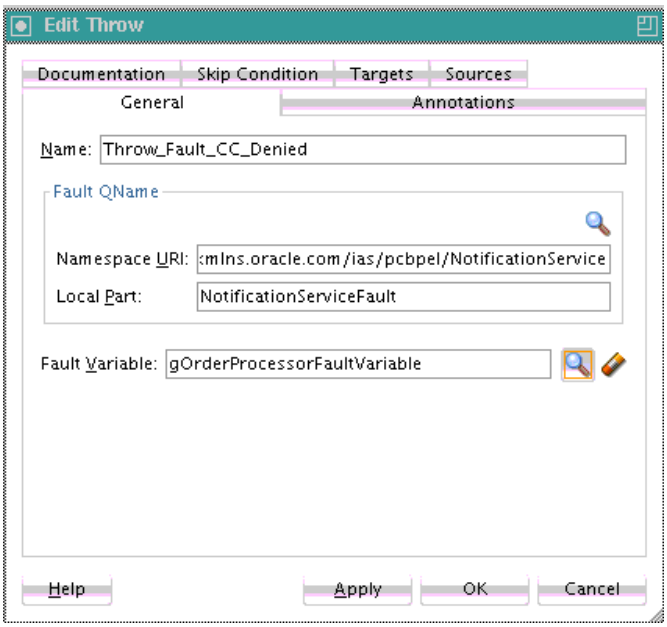
For more information about the terminate activity, see Stopping a Business Process Instance with the Terminate Activity in BPEL 1.1.

Throw Activity

This activity generates a fault from inside the business process.

Figure A-44 shows the Throw dialog in BPEL 2.0.

Figure A-44 Throw Dialog



For more information about the throw activity, see Throwing Internal Faults with the Throw Activity.

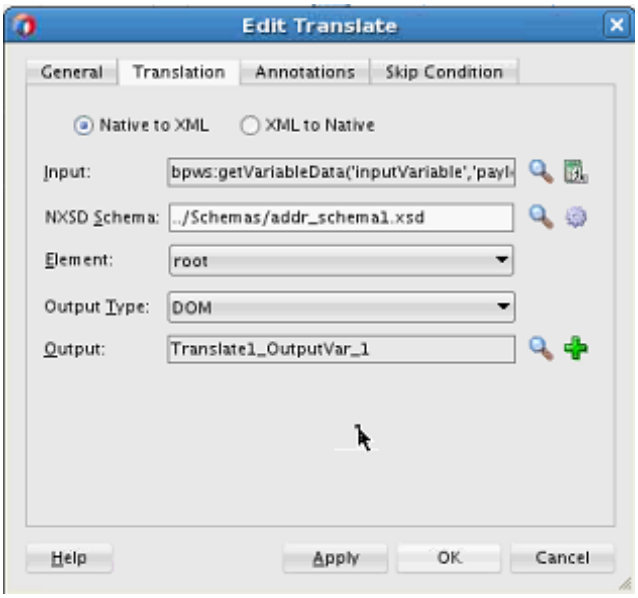
Translate Activity

This activity enables you to configure an inbound (with automatic use of the `doTranslateFromNative` function) translation or outbound (with automatic use of the `doTranslateToNative` function) translation.

- Inbound translation consists of native format to XML and opaque to XML.
- Outbound translation consists of XML to native format and large XML to an attachment in a directory.

This activity is supported in both BPEL 1.1. and 2.0. Figure A-45 shows the Translate dialog.

Figure A-45 Translate Dialog

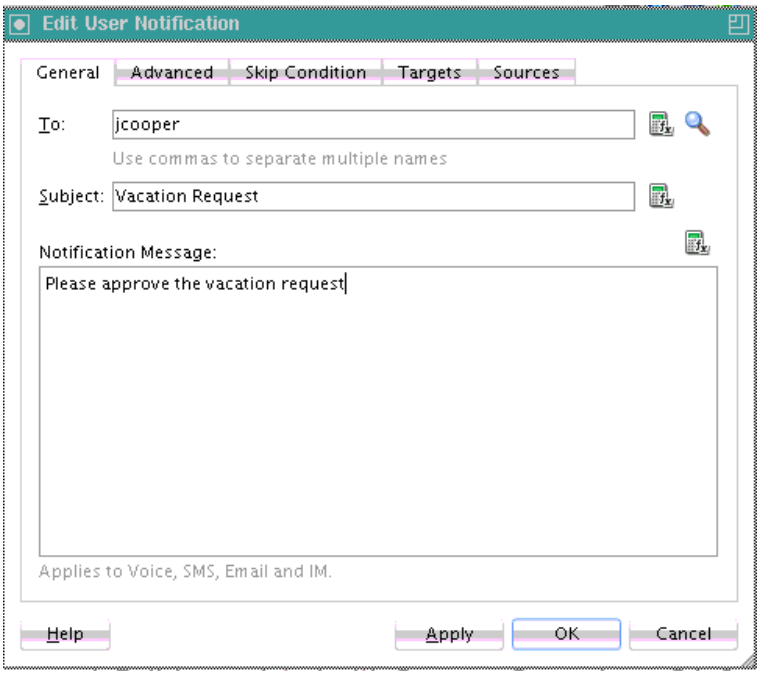


For more information, see Translating Between Native Data and XML.

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-46 provides details.

Figure A-46 User Notification Dialog



For more information about user notifications, see Allowing the End User to Select Notification Channels.

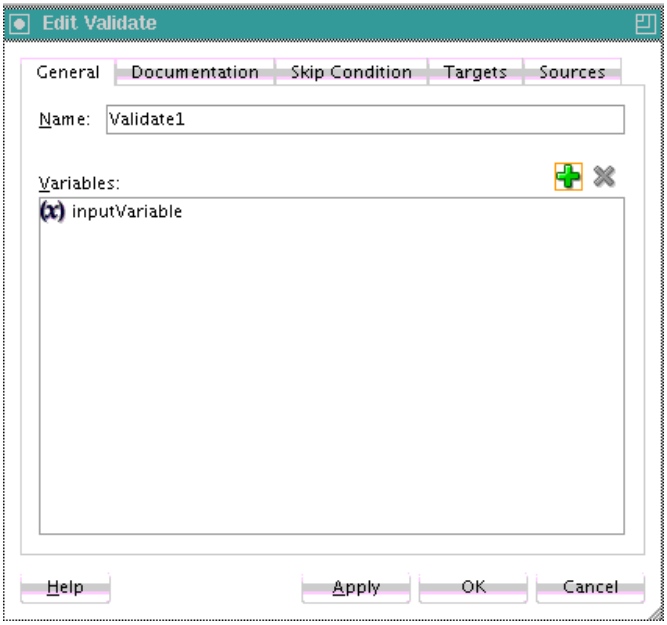
For more information about the Oracle User Messaging Service, see *Administering Oracle User Messaging Service* and *Developing Applications with Oracle User Messaging Service*.

Validate Activity

This activity enables you to validate variables in the list. The variables are validated against their XML schema.

Figure A-47 shows the Validate dialog in BPEL 2.0.

Figure A-47 Validate Dialog



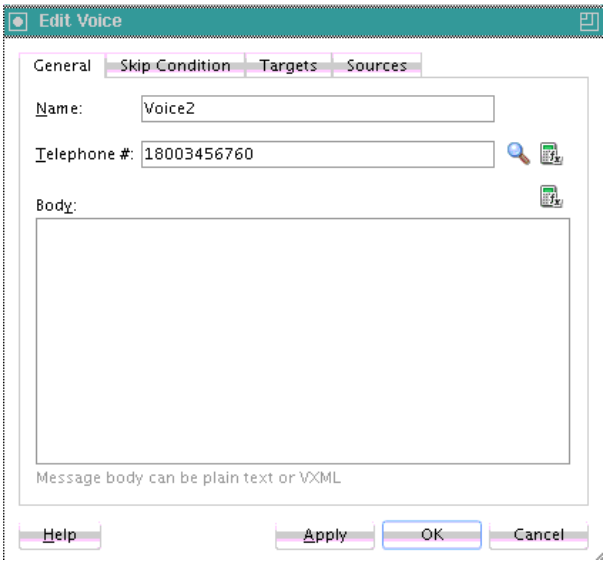
For more information about the validate activity, see Validating XML Data.

Voice Activity

This activity enables you to send a telephone voice notification about an event.

Figure A-48 shows the Voice dialog.

Figure A-48 Voice Dialog



For more information about the voice activity, see How to Configure the Voice Notification Channel.

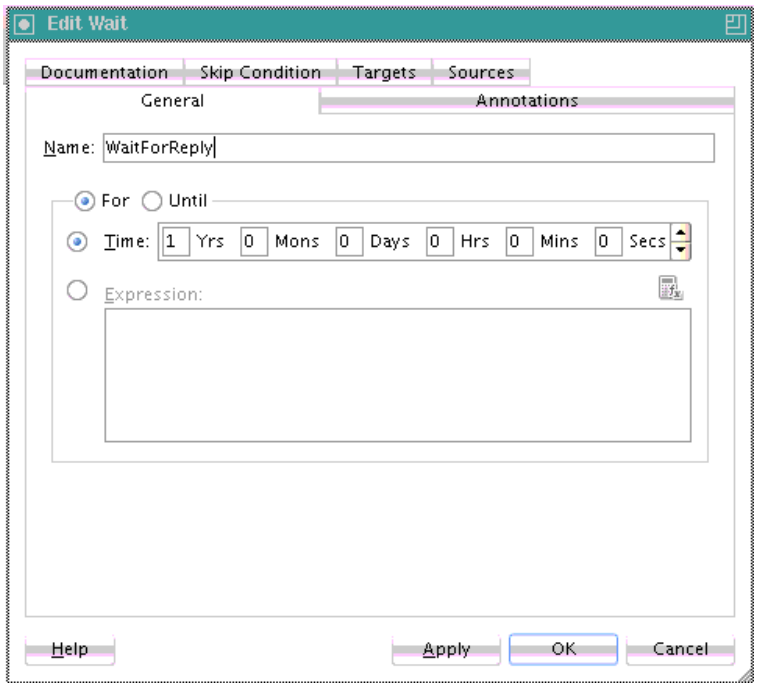
Wait Activity

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-49 shows the Wait dialog in BPEL 2.0.

Figure A-49 Wait Dialog



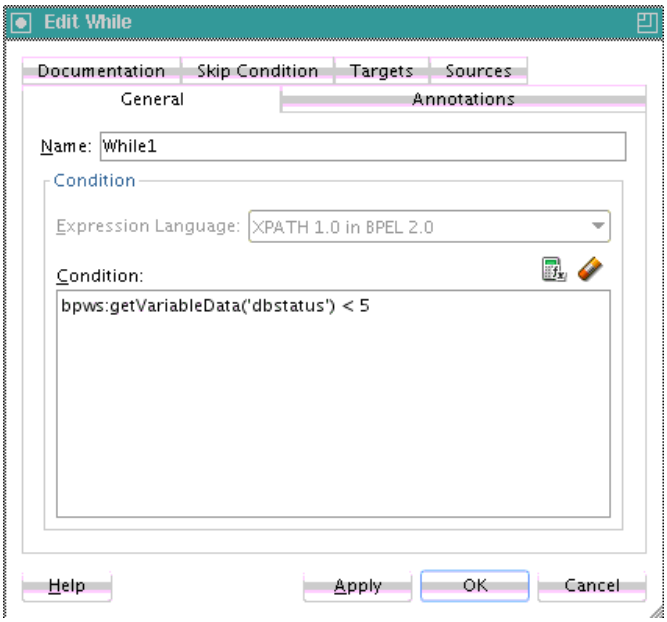
For more information about the wait activity, see Setting an Expiration Time with a Wait Activity .

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-50 shows the While dialog in BPEL 2.0. You can enter expressions in this dialog.

Figure A-50 While Dialog



For more information about the while activity, see Defining Conditional Branching with the While Activity.

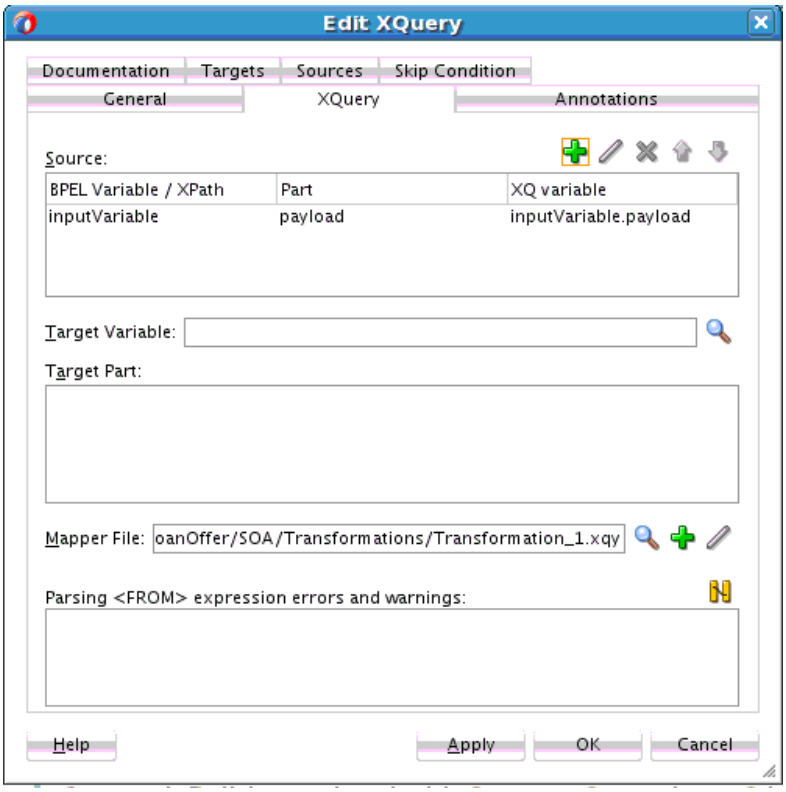
XQuery Transform Activity

This activity enables you to create an XQuery transformation that maps source elements to target elements (for example, incoming purchase order data into outgoing purchase order acknowledgment data).

Figure A-51 shows the XQuery dialog in BPEL 2.0. This dialog enables you to perform the following tasks:

- Define the source and target variables and parts to map.
- Specify the XQuery mapper file.
- Click the second icon (the **Add** icon) to the right of the **Mapper File** field to access the XQuery Mapper for creating a new XQuery file for graphically mapping source and target elements. Click the **Edit** icon (third icon) to edit an existing XQuery file.

Figure A-51 XQuery Dialog



For more information, see Creating Transformations with the XQuery Mapper.

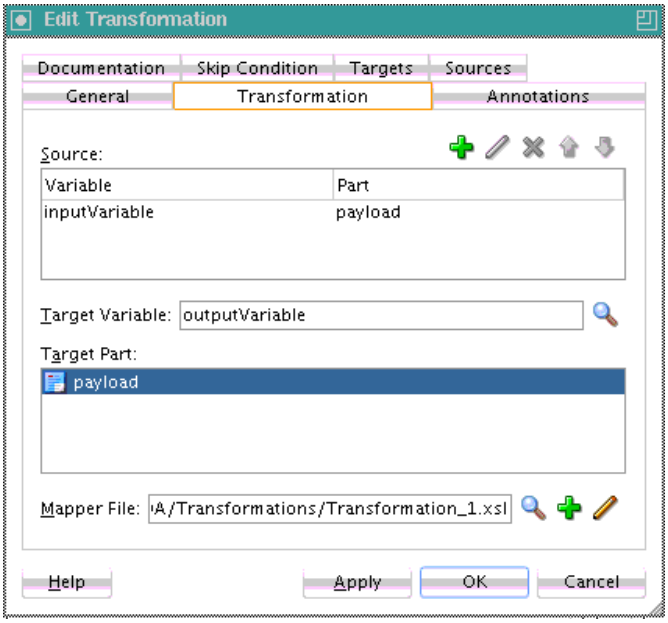
XSLT Transform Activity

This activity enables you to create an XSL transformation that maps source elements to target elements (for example, incoming purchase order data into outgoing purchase order acknowledgment data).

Figure A-52 shows the Transform dialog in BPEL 2.0. 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 Map Editor 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-52 Transform Dialog



For more information about the transform activity, see Creating Transformations with the XSLT Map Editor .

Introduction to BPEL Services

BPEL processes can communicate with web-based applications and clients through SOAP web services, Oracle ADF Business Component (BC) services, JCA adapters, Oracle B2B services, Oracle Healthcare services, Oracle Business Activity Monitoring 11*g*, HTTP binding, direct binding, EJB services, REST adapters, Oracle E-Business Suite, JDE World, SAP, cloud adapters, and partner links.

To access BPEL services:

1. In the Components window of Oracle BPEL Designer, expand **BPEL Services** to display the services.
2. Drag and drop the service to the appropriate swimlane. Table A-2 lists the available services and provides references to documentation that describes these services.

Table A-2 BPEL Services

BPEL Service	For More Information, See...
ADF-BC services	ADF-BC Services
AQ adapter	• AQ Adapter • *Understanding Technology Adapters*
Oracle B2B	• Oracle B2B • *User's Guide for Oracle B2B*
Oracle Business Activity Monitoring (BAM) 11*g*	• Oracle BAM 11g Adapter • *Monitoring Business Activity with Oracle BAM*

Table A-2 (Cont.) BPEL Services

BPEL Service	For More Information, See...
Coherence Cache	• Coherence Adapter • *Understanding Technology Adapters*
Database adapter	• Database Adapter • *Understanding Technology Adapters*
Direct binding service	• Direct Binding Adapter • Using Direct Binding to Invoke Composite Services • *Java API Reference for Infrastructure Management*
Oracle E-Business Suite adapter	• Oracle E-Business Suite Adapter • *Oracle E-Business Suite Adapter User's Guide*
EJB service	• EJB Adapter • Integrating Enterprise JavaBeans with Composite Applications
File adapter	• File Adapter • *Understanding Technology Adapters*
FTP adapter	• FTP Adapter • *Understanding Technology Adapters*
Healthcare adapter	• Oracle Healthcare Adapter • olink:HFPUG1732*Healthcare Integration User's Guide for Oracle SOA Suite*
HTTP binding	HTTP Binding Service
JDE World adapter	`http://www.oracle.com/technetwork/middleware/adapters/documentation/index.html`
JMS adapter	• JMS Adapter • *Understanding Technology Adapters*
LDAP	• LDAP Adapter • *Using Oracle Managed File Transfer*
Oracle MFT	• Oracle MFT • *Using Oracle Managed File Transfer*
MQ adapter	• MQ Adapter • *Understanding Technology Adapters*
REST service	Integrating REST Operations in SOA Composite Applications
Cloud adapters	Cloud Adapters
SAP adapter	`http://www.oracle.com/technetwork/middleware/adapters/documentation/index.html`
SOAP	• Adding Service Binding Components • SOAP Web Services
Socket adapter	• Socket Adapter • *Understanding Technology Adapters*

Table A-2 (Cont.) BPEL Services

BPEL Service	For More Information, See...
Third-party adapter	• Third-Party Adapter • *Understanding Technology Adapters*
Oracle User Messaging Service	• Oracle User Messaging Service Adapter • *Understanding Technology Adapters*

B

XPath Extension Functions

This appendix describes the XPath extension functions that are displayed in the Expression Builder dialog in Oracle JDeveloper. It also describes how to build XPath expressions in the Expression Builder and how to create user-defined 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:

For additional information about XPath functions, visit the following URL:

`http://www.w3.org`

Advanced Functions

This section describes the advanced functions.

batchProcessActive

This function returns the number of active processes in the batch.

Signature:

```
ora:batchProcessActive(String rootId, String processId)
```

Arguments:

- `rootId`: The ID of the root.
- `processId`: The ID of the process.

Property IDs:

- `namespace-uri:http://schemas.oracle.com/xpath/extension`
- `namespace-prefix: ora`

batchProcessCompleted

This function returns the number of completed processes in the batch.

Signature:

```
ora:batchProcessCompleted(String rootId, String processId)
```

Arguments:

- `rootId`: The ID of the root.
- `processId`: The ID of the process.

Property IDs:

- `namespace-uri:http://schemas.oracle.com/xpath/extension`
- `namespace-prefix: ora`

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 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`

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`

createDelimitedString

This function creates a delimited string from the passed-in arguments.

Signature:

```
ora:createDelimitedString(delimiter as string, nodeList)
```

Arguments:

- `delimiter as string`: The character that separates the items in the input string (for example, a comma or a semicolon).

- `nodeList`: Provides an ordered collection of nodes.

Property IDs:

- `namespace-uri:`
- `namespace-prefix:ora`

createEssParameter

This function creates a parameter for a job in Oracle Enterprise Scheduler.

Signature:

```
ess:createEssParameter(dataType,name,scope,value)
```

Arguments:

- `dataType`
- `name`
- `scope`
- `value`

Property IDs:

- `namespace-uri:`
- `namespace-prefix:ess`

For more information about Oracle Enterprise Scheduler, see *Developing Applications for Oracle Enterprise Scheduler*.

doStreamingTranslate

This function translates using the streaming XPath APIs. It uses batching so that the transformation engine does not materialize the result of the 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($in.request/inp1:request/
inp1:sourceAttachmentElement,$in.request/inp1:request/
inp1:streamingcontext, 'ATTACHMENT', $in.request/inp1:request/
inp1:targetAttachmentElement)
```

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:source),'xsd/address_csv.xsd','Root-Element','DOM')
```

doTranslateToNative

This function translates the input DOM to a string or attachment. The targetType can be a 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 the following example:

  ```
  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 the following example:

  ```
  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/
address_csv.xsd','Root-Element','STRING')
```

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`

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`

For more information about this function, see Generating Functionality Equivalent to an Array of an Empty Element.

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`

get-content-from-file-function

This function parses the file in the specified native format. Use this function when designing assign activities in BPEL processes.

Signature:

```
oraext:get-content-from-file-function(fileName, nxsdTemplate?,
nxsdRoot?)
```

Example:

```
oraext:get-content-from-file-function("file:/c:/Ftab.txt",
"file:/c:/Ftab_1.xsd","root")
```

Arguments:

- `fileName`: The name of the file.
- `nxsdTemplate`: The native XSD (NXSD) template for the output.
- nxsdRoot: The NXSD root.

Property IDs:

- `namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix`: `oraext`

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`

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`

For more information, see Reading and Encoding SOAP Attachment Content.

getAttachmentProperty

Gets a SOAP attachment property from an `href` that is stored in `varName`, `partName`, and `query`.

Signature: `ora:getAttachmentProperty(propertyName, 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`

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`

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`

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`

getCompositeInstanceID

Note:

This function is deprecated in 12*c* Release 1 (12.1.3) and is not displayed in the Expression Builder.

This function returns the composite 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`

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`

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`

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`

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`

For more information, see Getting Fault Details with the getFaultAsString XPath Extension Function.

getFaultAsXML

This function returns the fault as an XML element.

Signature:

`ora:getFaultAsXML()`

Arguments:

There are no arguments for this function.

Property IDs:

- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix:ora

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

getMilestoneName

This function returns the milestone name.

Signature:

```
ora:getMilestoneName
```

Arguments:

There are no arguments for this function.

Property IDs:

- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

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

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`

getRevision

This function does not take any arguments and returns the current revision of the composite from which it is invoked.

Signature:

```
ora:getRevision
```

Arguments:

There are no arguments for this function.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

getTaskReminderDuration

This function computes the next reminder to be sent for the task.

Signature:

```
ora:getTaskReminderDuration(taskId)
```

Argument:

- `taskId`: The task ID of the task.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix: ora`

instanceOf

This function extracts arbitrary values from BPEL variables.

Signature:

```
ora:instanceOf(an_xpath_expression, 'typeQName')
```

Arguments:

- `an_xpath_expressio`n: 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

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

parseEscapedXML

This function parses a string to a DOM.

Note:

This function is also displayed for selection under the **BPEL XPath Extension Functions** option.

Signature:

```
oraext: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`: `oraext`

For more information about this function, see How To Convert from a String to an XML Element.

parseXML

This function parses a string to a DOM element.

Signature:

```
oraext:parseXML(contentString)
```

Arguments:

- `contentString`: The string that this function parses to a DOM element.

Property IDs:

- `namespace-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix`: `oraext`

processScalableDocumentToNative

This function transforms the scalable document directly to the output stream.

Signature:

```
ora:processScalableDocumentToNative(template, input,
outputFilePath, nxsd, root, batchsize, properties)
```

Arguments:

- `template`
- `input`
- `outputFilePath`
- `nxsd`
- `root`
- `batchsize`
- `properties`

processXSLTAttachmentFromNativeToNative

This function translates the inbound native data (for example, comma-separated value to XML) and then applies the user-supplied XSL to the translated content. The result of the XSL transformation is then translated to a native file (for example. comma-separated value). The input to this XPath function can either be an attachment or `href`. It uses batching so that the transformation engine does not put the result of the transformation into memory. Therefore, it can handle arbitrarily large payloads of the order of gigabytes. The XPath function translates the inbound native data to XML, runs the transformation on the XML, and then translates the transformed XML to native format.

Signature:

```
ora:processXSLTAttachmentFromNativeToNative(template, input
href, output href, input nxsd path, input root element name,
output nxsd path, output root element name. batch size)
```

processXSLTAttachmentFromNativeToStream

This function translates the inbound native data (for example, customer-separated value to XML) and then applies the user-supplied XSL to the translated content. The output of the transformation is streamed to the output file.The input to this XPath function can either be an attachment or `href`. It uses batching so that the transformation engine does not put the result of the 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`.

Signature:

```
ora:processXSLTAttachmentFromNativeToStream(template,input href,
output href, input nxsd path, nxsd root element name,
batchsize,properties)
```

processXSLTAttachmentToNativeStream

This function transforms the inbound XML by applying the user-supplied XSL and then translates the transformed XML into a native file (for example, comma-separated value). The input to this XPath function can either be an attachment or `href`. It uses batching so that the transformation engine does not put the result of the transformation into memory. Therefore, it can handle arbitrarily large payloads of the order of gigabytes. This function first transforms the incoming XML data by applying the XSL and then translates the transformed XML into native data.

Signature:

```
ora:processXSLTAttachmentToNativeStream(template, input href,
output href, nxsd schema, nxsd root element, batch size)
```

processXSLTAttachmentToStream

This function directly streams the result of XSLT transformation to the output file. The input to this XPath function can either be an attachment or `href`. It uses batching so that the transformation engine does not put the result of the 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`.

Signature:

```
ora:processXSLTAttachmentToStream(template, input href, output
href, batchsize, properties)
```

processXSLTForScalableDocument

This function returns a scalable document after an XSLT transformation.

Signature:

```
ora:processXSLTForScalableDocument(template, input, batchsize,
properties)
```

setCompositeInstanceTitle

This function sets the composite instance title and returns it.

Signature:

```
ora:setCompositeInstanceTitle(title)
```

Arguments:

- `title`: The composite instance title.

BPEL Extension Functions

This section describes the BPEL extension functions.

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 2.0 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
`bpel:doXslTransform`	No	Yes

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`

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 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`

selectionFailure Fault is Thrown if the Result Node Set is a Size Other Than One During Execution

According to the , 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 the following code does not work if there are multiple entries of `product` elements under `StoreRequest`; this causes a `selectionFailure` fault to be thrown:

```
count(bpws:getVariableData('inputVariable',
 'payload','/ns2:StoreRequest/ns2:product'))
```

To make this work, change the syntax to the following:

```
"count($inputVariable.payload/ns2:product)"
```

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`

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`

doXslTransform (For BPEL 2.0)

This function returns the result of XSLT transformation with multiple sources.

Note:

If the input is meant to be an XML document, call `ora:getOwnerDocument` to wrap the input or use function `ora:doXSLTransformForDoc` instead of this function.

Signature:

```
bpel:doXslTransform(template,input, [paramQName, paramValue]*)
```

BPEL XPath Extension Functions

This section describes the BPEL XPath extension functions.

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`

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.

The following example shows the format of the `directories.xml` file:

```
<?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>
```

The following shows an example of the `directories.xml` file:

```
<?xml version="1.0" ?>
<directories>
<directory name='people'>
<property
name="java.naming.provider.url">ldap://myhost.us.example.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`

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`

doXSLTransform

This function implements the 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`

doXSLTransformForDoc

This function is a complementary XPath function to `doXSLTransform()`. It aims to perform the transformation when the XSLT template matches the document.

The following example shows the `doXSLTransformForDoc` function:

```
<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 Using XPath Functions to Write Large XSLT/XQuery Output to a File System.

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`

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`

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`

getConfigProperty

This function gets the component property value.

Signature:

```
ora:getConfigProperty(propertyName)
```

Argument:

- `propertyName`: The property name.

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`

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`

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`

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 How to Assign a Date or Time.

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`

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`

getElement

This function returns an element using an `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`

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`

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`

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`

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`

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`

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`

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`

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`

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`

listUsers

This function returns a list of LDAP users.

Signature:

```
ldap:listUsers('directoryName',filter')
```

Arguments:

- `directoryName`: The directory name specified in the `directories.xml` file. For information about the `directories.xml` file, see authenticate.
- `filter`: The filter expression to use for the search; this value cannot be `null`.

Returns:

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');
```

The following provides an example of the 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>
```

lookupUser

This function returns LDAP user information.

:**Signature**:

`ldap:lookupUser('directoryName','userId')`

Arguments:

- `directoryName`: The directory name specified in the `directories.xml` file. For information about the `directories.xml` file, see authenticate.
- `userId`: The user ID to be searched.

Returns:

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');`

The following provides an example of the output:

```
<user dn="" xmlns="http://schemas.oracle.com/bpel/ldap">
<ou>people</ou>
<objectclass>organizationalUnit</objectclass>
<objectclass>top</objectclass>
<orclguid>8ABB9BA06FDD11DEBF9A7F3D47003274</orclguid>
</user>
```

parseEscapedXML

This function parses an XML string to an XML element.

Note:

This function is also displayed for selection under the **Advanced Functions** option.

Signature:

`ora:parseEscapedXML(xmlString)`

Arguments:

- `xmlString`: The string that this function parses to a DOM.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

For more information about this function, see How To Convert from a String to an XML Element.

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`

processXQuery10

This function returns the result of an XQuery 1.0 transformation.

Signature: `ora:processXQuery10(<path to xquery> [, <xquery external variable name>, <value>]*)`

processXQuery2004

This function returns the result of an XQuery 2004 transformation.

Signature: `ora:processXQuery2004(template,context?)`

processXSLT

This function returns the result of an XSLT transformation using the Oracle XDK XSLT processor.

The following example shows the 12*c* 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>
```

Signature:

- 12*c* version of the signature:

  `ora: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 a BPEL process, you use the properties to process the XSLT function.

Property IDs:

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora` (for 12*c*)

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 Using XPath Functions to Write Large XSLT/XQuery Output to a File System.

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`

For more information, see Sending Attachment Streams.

readBinaryFromFileWithMimeHeaders

This function returns the content of a binary file with MIME headers.

Signature:

```
ora:readBinaryFromFileWithMimeHeaders(fileName, contentId,
contentType, contentDisposition, contentTransferEncoding,
contentDescription, contentLanguage)
```

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 the following example:

  ```
  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
```

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 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`.

Returns:

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');
```

The following provides an example of the 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>
```

toCDATA

This function returns a DOM node as a CDATA section.

Signature:

`ora:toCDATA(node)`

tryToCastToBoolean

This function returns a boolean value if the input is a string of `true`, `false`, `1`, or `0`.

Signature:

`ora:tryToCastToBoolean(string)`

Argument:

- `string`: String value to attempt to convert to a boolean value.

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`

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`

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`

Conversion Functions

This section describes the conversion functions.

boolean

This function converts the input to a boolean. A number is true only if it is neither positive or negative zero or NaN. A node-set is true only if it is nonempty. A string is true only if its length is nonzero.

Signature:

```
boolean(input as any)
```

Arguments

- `input as any`: Any value.

For example:

`boolean('false')` returns `true`.

Property IDs:

- `namespace-uri:`

number

This function converts the input to a number. A string that consists of optional white space, followed by an optional minus sign, followed by a number, followed by white

space is converted to the IEEE 754 number that is nearest (according to the IEEE 754 round-to-nearest rule) to the mathematical value represented by the string. Any other string is converted to a NaN. A boolean true is converted to 1. A boolean false is converted to 0. A node-set is first converted to a string as if by a call to the string function and then converted in the same way as a string parameter.

Signature:

```
number(input as string or boolean or node-set)
```

Arguments

- `input as string or boolean or node-set`: Value to convert.

For example:

`number('12.3')` returns `12.3`.

string

This function converts an object to a string.

Signature:

```
string(input as any)
```

Arguments

- `input as any`: The object to convert.

For example:

`string(12.3)` returns `'12.3'`.

Property IDs:

- `namespace-uri:`
- `namespace-prefix:`

DVM Functions

This section describes the domain value map (DVM) functions.

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 dvm:lookupValue.

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 dvm:lookupValue1M.

Database Functions

This section describes the database functions.

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
```

You execute it 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`

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`

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
```

You execute it 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`

Date Functions

This section describes the date functions.

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

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

current-dateTime

This function returns the current datetime value in the ISO format of CCYY-MM-DDThh:mm:ss.sTZD (where s denotes the time in milliseconds).

For example, if the time is 6 hours, 17 minutes, 15 seconds, 125 milliseconds in the evening (PM) of May 12, 2004 in time zone Z, current-dateTime returns a value of:

```
2004-05-12T18:17:15.125Z
```

If com.oracle.soa.xpath.datetimeWithoutMillis is set to true in the setDomainEnv file, this function returns the current datetime value in the following format (where ss denotes the time in seconds):

```
CCYY-MM-DDThh:mm:ss.TZD
```

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

To display the datetime value in seconds:

1. Open the following file:
   - On UNIX operating systems, open $MIDDLEWARE_HOME/user_projects/domains/*domain_name*/bin/setDomainEnv.sh.

- On Window operating systems, open *MIDDLEWARE_HOME*\user_projects\domains*domain_name*\bin\setDomainEnv.bat.

2. Add com.oracle.soa.xpath.datetimeWithoutMillis with a value of true in the JAVA_OPTIONS section. For example, JAVA_OPTIONS is currently set as follows:

```
JAVA_OPTIONS="${JAVA_OPTIONS} ${JAVA_PROPERTIES}
-Dwlw.iterativeDev=${iterativeDevFlag} -Dwlw.testConsole=${testConsoleFlag}
-Dwlw.logErrorsToConsole=${logErrorsToConsoleFlag} "
```

   After modification, JAVA_OPTIONS appears as follows:

```
 JAVA_OPTIONS="${JAVA_OPTIONS} ${JAVA_PROPERTIES}
-Dwlw.iterativeDev=${iterativeDevFlag} -Dwlw.testConsole=${testConsoleFlag}
-Dwlw.logErrorsToConsole=${logErrorsToConsoleFlag}
-Dcom.oracle.soa.xpath.datetimeWithoutMillis=true"
```

3. Restart the server.

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

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

format-dateTime

This function returns the formatted string of `dateTime` using the format provided. For examples of date and time formatting strings, see the W3C XSL Transformations documentation; for example, `[Y0001]-[M01]-[D01]`.

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`

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`

minutes-from-dateTime

This function returns the minutes 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`

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`

seconds-from-dateTime

This function returns the seconds 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`

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

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

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

Identity Service Functions

This section describes the identity service functions.

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`

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.

  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`

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`

getManagerFromManagementChain

This function gets the management chain for a given user based on `upToUserName`, `upToTitle`, and `upToLevel`. If the user does not exist or if there is no manager for the user, it returns null. Regular expressions can be used in `upToTitle` and `upToUser` parameters.

Signature:

```
ids:getManagerFromManagementChain()
```

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`

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`

getUserProperty

This function returns the property of the user. If the user does not exist, it returns `null`. Use custom attributes if the desired attribute does not exist.

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.

  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`

For more information, see How to Select Email Addresses and Telephone Numbers Dynamically.

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`

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.

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`

isUserInAppRole

This function verifies if a user has a specific application role.

Signature:

```
ids:isUserInAppRole(userName, appRoleName, appName, realmName)
```

Arguments:

- `userName`: String or element containing the user whose participation in the role must be verified.
- `appRoleName`: The application role name.
- `appName`: The application name (for example, `OracleBPMProcessRolesApp`, `OracleBPMComposerRolesApp`, and so on).
- `realmName`: The realm name. This is optional. If not specified, the default realm is assumed. This function returns a boolean true or false.

isUserInRole

This function verifies if a user has a specific 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`

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`

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`

Logical Functions

This section describes the logical function.

and

This function returns true if both parameters evaluate to true. Otherwise, it returns false.

Signature:

```
a-boolean and another-boolean
```

Arguments:

- `a-boolean`: One boolean value to evaluate.
- `another-boolean`: The other boolean value to evaluate.

equals

This function returns true if the two parameters are equal. Otherwise, it returns false.

Signature:

```
parameter1 = parameter2
```

Arguments:

- `parameter1`: One parameter to evaluate.
- `parameter2`: The other parameter to evaluate.

false

This function returns a boolean value of false.

Signature:

```
false()
```

greater

This function returns true if the first parameter is greater than the second parameter. Otherwise, it returns false.

Signature:

```
parameter1 > parameter2
```

Arguments:

- `parameter1`: First parameter to evaluate.
- `parameter2`: Second parameter to evaluate.

greater equals

This function returns true if the first parameter is greater than or equal to the second parameter. Otherwise, it returns false.

Signature:

```
parameter1 >= parameter2
```

less

This function returns true if the first parameter is less than the second parameter. Otherwise, it returns false.

Signature:

```
parameter1 < parameter2
```

Arguments:

- `parameter1`: First parameter to evaluate.
- `parameter2`: Second parameter to evaluate.

less equals

This function returns true if the first parameter is less than or equal to the second parameter. Otherwise, it returns false.

Signature:

```
parameter1 <= parameter2
```

Arguments:

- `parameter1`: First parameter to evaluate.
- `parameter2`: Second parameter to evaluate.

not

This function returns the negation of the parameter.

Signature:

```
unobtainable as boolean)
```

Argument:

- `input as boolean`: The value to evaluate.

not equals

This function returns true if the two parameters are not equal. Otherwise, it returns false.

Signature:

```
parameter1!= parameter2
```

Arguments:

- `parameter1`: First parameter to evaluate.
- `parameter2`: Second parameter to evaluate.

or

This function returns true if either parameter evaluates to true. Otherwise, it returns false.

Signature:

```
a-boolean or another-boolean
```

Arguments:

- `a-boolean`: First parameter to evaluate.
- `another-boolean`: Second parameter to evaluate.

true

This function returns a boolean value of true.

Signature:

`true()`

Property IDs:

- `namespace-uri`:
- `namespace-prefix`:

Mathematical Functions

This section describes the mathematical functions.

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`

add

This function adds two numbers.

Example:

2 + 2 = 4

ceiling

This function returns the smallest (closest to negative infinity) number that is not less than the input number and is an integer.

Example:

`ceiling(1.6)` returns `2.0`.

count

This function returns the number of nodes in the input node set.

Example:

```
count(inputNodeSet as node-set)
```

Argument:

- `inputNodeSet`: The input node set.

divide

This function returns the first number divided by the second number.

Example:

`2 div 2 = 1`

floor

This function returns the largest (closest to positive infinity) number that is not greater than the input number and is an integer.

Signature:

`floor(1.6)` returns `1.0`

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`

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`

mod

This function returns the remainder from a truncating division.

Example:

`5 mod 2` returns `1`

multiply

This function multiplies two numbers.

Example:

`2 * 2 = 4`

round

This function returns the number that is closest to the input number and is an integer. If there are two numbers, the one that is closest to positive infinity is returned.

Example:

`round(1.5)` returns `2.0`.

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`

subtract

This function subtracts the second number from the first number.

Example:

```
2 - 2 = 0
```

sum

This function returns the sum of all nodes in numbers.

Signature:

```
sum(numbers as node-set-set)
```

Argument:

- `numbers as node-set-set`: Total number of node sets.

Property IDs:

- `namespace-uri`:
- `namespace-prefix`:

unary

This function multiplies a number by `-1`.

Signature:

```
-(-1) = 1
```

Node Set Functions

This section describes the node set functions.

last

This function returns the context size.

Signature:

```
last()
```

local-name

This function returns the local part of the name of a node.

Signature:

```
local-name([inputNodeSet as node-set])
```

Arguments:

- `inputNodeSet as node-set`: The name of the node set.

name

This function returns the `QName` of a node.

Signature:

```
name([inputNodeSet as node-set])
```

Argument:

- `inputNodeSet as node-set`: The name of the node set.

namespace-uri

This function returns the URI namespace of a node.

Signature:

```
namespace-uri([inputNodeSet as node])
```

Argument:

- `inputNodeSet as node-set`: The name of the node set.

position

This function returns the context position.

Signature:

```
position()
```

union

This function computes the union of its operands, which must be node sets.

Signature:

node-set | node-set

String Functions

This section describes the string functions.

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

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. Table B-2 provides details.

Table B-2 Values Returned

This Function Returns...	If...
-1	inputString lexicographically precedes the compareString.
0	Both inputString and compareString are equal.
1	inputString lexicographically follows the compareString.

Example:

oraext: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: oraext

concat

This function returns the concatenation of its string parameters.

Signature:

```
concat(string1 as string, string2 as string, ...)
```

Arguments:

- string1: String value to concatenate.
- string2: String value to concatenate.

contains

This function returns true if inputString contains searchString. Otherwise, it returns false.

Signature:

```
contains(inputString as string,searchString as string)
```

For example:

`contains('Michael Kay','Michael')` returns `true`.

create-delimited-string

This function returns a delimited string created from a `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`

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`

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 use in the formatted output.

Property IDs:

- `namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix`: `oraext`

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`

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`

index-within-string

This function returns the zero-based index of the first occurrence of `searchString` within the `inputString`.

This function returns `-1` if `searchString` is not found.

Example:

`oraext:index-within-string('ABCABC, 'B')` returns `1`

Signature:

```
oraext:index-within-string(inputString as string, searchString as string)
```

Arguments:

- `inputString`: The string of data to be searched.
- searchString: The string for which the function searches in `inputString`.

Property IDs:

- `namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix`: `oraext`

last-index-within-string

This function returns the zero-based index of the last occurrence of `searchString` within `inputString`.

This function returns `-1` if `searchString` is not found.

Example:

`oraext:last-index-within-string('ABCABC', 'B')` returns 4

Signature:

```
oraext:last-index-within-string(inputString as string, searchString as string)
```

Arguments:

- `inputString`: The string of data to be searched.
- searchString: The string for which the function searches in the `inputString`.

Property IDs:

- `namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix`: `oraext`

left-trim

This function returns the value of `inputString` after removing all the leading white spaces.

Example:

`oraext:left-trim(' account ') returns 'account '`

Signature:

`oraext:left-trim(inputString)`

Arguments:

- `inputString`: The string to be left-trimmed.

Property IDs:

- `namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix`: `oraext`

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`

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`

normalize-space

This function returns the input string with white space normalized by stripping leading and trailing white space and replacing sequences of white space characters with a single space.

Signature:

```
normalize-space([inputString as string])
```

Arguments:

- `inputString`: The input string.

For example:

`normalize-space(' book title ')` returns `'book title'`.

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`

starts-with

This function returns true if the input string starts with a search string. Otherwise, it returns false.

Signature:

```
starts-with(inputString as string,searchString as string)
```

Arguments:

- `inputString`: The input string.
- `searchString`: The search string.

For example:

`starts-with('data type','data')` returns `true`.

string-length

This function returns the number of characters in the input string.

Signature:

```
string-length([inputString as string])
```

Argument:

- `inputString`: The input string.

For example,

`string-length('xml')` returns 3.

Property IDs:

- `namespace-uri`:
- `namespace-prefix`:

substring

This function returns the substring of the input string starting at the position specified in the starting location with the length specified in length.

Signature:

```
substring(inputString as string,startingLoc as number,[length as number])
```

Arguments:

- `inputString`: The input string.
- `startingLoc`: The starting location.
- `length as number`: The length as a number.

For example:

`substring('12345',2)` returns `'2345'`.

substring-after

This function returns the substring of the input string that follows the first occurrence of the search string, or the empty string if the input string does not contain the search string.

Signature:

```
substring-after(inputString as string,searchString as string)
```

Arguments:

- `inputString`: The input string.
- `searchString`: The string for which to search.

For example,

`substring-after('1999/04/01','/')` returns `'04/01`.

substring-before

This function returns the substring of the input string that precedes the first occurrence of the search string or the empty string if the input string does not contain the search string.

Signature:

```
substring-before(inputString as string,searchString as string)
```

Arguments:

- `inputString`: The input string.
- `searchString`: The string for which to search.

For example:

`substring-before('1999/04/01','/')` returns `'1999'`.

translate

Signature:

```
translate(inputString as string,fromString as string,toString as
string)
```

Arguments:

- `inputString`: The input string.
- `fromString`: The from string.
- `toString`: The to string.

For example,

`translate('--aaa--','abc-','ABC')` returns `'AAA'`.

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`

Workflow Service Functions

This section describes the workflow service functions.

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`

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`

dynamicTaskAssign

This function selects an assignee of the specified type from the input, using the specified pattern, in the context of the current task.

This function can only be used in the context of a human task.

Signature:

```
hwf:dynamicTaskAssign(patternName, participants,
inputParticipantType, targetAssigneeType, isGlobal,
invocationContext, parameter1, parameter2, ..., parameterN)
```

Arguments:

- `patternName`: (Mandatory) Name of the pattern to use. The patterns `ROUND_ROBIN`, `LEAST_BUSY`, and `MOST_PRODUCTIVE` are automatically provided. It is possible to configure the SOA server with custom patterns.
- `participants`: (Mandatory) The participant or participants from which to select the assignee. This can be a string or element containing a participant name or a comma-separated list of participant names, or a set of elements containing participant names or comma-separated lists of participant names. Participants must all be of the same type.
- `inputParticipantType`: (Mandatory) The type of the input participants (`user`, `group`, or `application_role`).
- `targetAssigneeType`: (Mandatory) The type of assignee to select (`user`, `group`, or `application_role`). The value must match the context in which the function is used (for example, it must be a user if dynamically selecting an owner user. Note that if `inputParticipantType` is the user, the only valid value here is the user.
- `isGlobal`: A boolean value that indicates to access the pattern using tasks of all types or tasks of the same type as the current task. This is optional. It defaults to false.
- `invocationContext`: The string to uniquely identify where this function is used. If not specified, a default context is assigned.
- `parameter`*`N`* : Some dynamic assignment patterns enable parameters to be specified. The parameter values can be specified as name-value pairs, using an `?=?` character as a delimiter (for example, `?TIME_PERIOD=7?`).

Examples:

```
hwf:dynamicTaskAssign(?LEAST_BUSY?,?jcooper,jstein,mtwain?,?
user?,?user?,?true?,?ErrorAssignee?)
```

```
hwf:dynamicTaskAssign(?ROUND_ROBIN?,?LoanAgentGroup?,?group?,?
user?,?false?,?OwnerUser?)
```

```
hwf:dynamicTaskAssign(?MOST_PRODUCTIVE?,task:task/task:payload/
task:users,?user?,?user?,?false?,?OwnerUser?,?TIME_PERIOD=7?)
```

```
hwf:dynamicTaskAssign(?LEAST_BUSY?,?DeveloperRole?,?
application_role?,?group?)
```

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`

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`

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`

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`

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`

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`

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`

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`

XREF Functions

This section describes the cross reference (XREF) functions.

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`

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,col
umnName,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 About the xref:lookupXRef Function.

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,c
olumnName,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 About the xref:lookupXRef1M Function.

markForDelete

This function deletes a value in a cross-reference table. The row, containing the column value passed to the function, is deleted from the `XREF_DATA` table and moved to the `XREF_DELETED_DATA` table. 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 that contains the value to be deleted.
- `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 How to Delete a Cross Reference Table Value.

populateLookupXRefRow

This function populates the column value in the cross-reference table (XREF) in which the reference column has the reference value. Depending on the mode, the reference value may also be populated. Unlike the `xref:populateXRefRow` function, the `xref:populateLookupXRefRow` function does not throw a unique constraint violation error when records with the same ID are added simultaneously. Instead, it behaves as a lookup and returns the existing source value that caused the error and does not stop the processing flow. Use this function to resolve any concurrency issues that can arise when using the `xref:populateXRefRow` function.

Signature:

```
xref:populateLookupXRefRow(xrefLocation as string,
referenceColumnName as string, referenceValue as string,
columnName as string, value as string, mode as string)
```

For example:

```
xref:populateLookupXRefRow("C:\xrefs\customer-id.xref", "Oracle
System" , "ORCL_100", "SAP System", "SAP_001", "ADD")
```

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,referenceValu
e,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 About the xref:populateXRefRow Function.

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,referenceVa
lue,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 About the xref:populateXRefRow1M Function.

Building XPath Expressions in the Expression Builder in Oracle JDeveloper

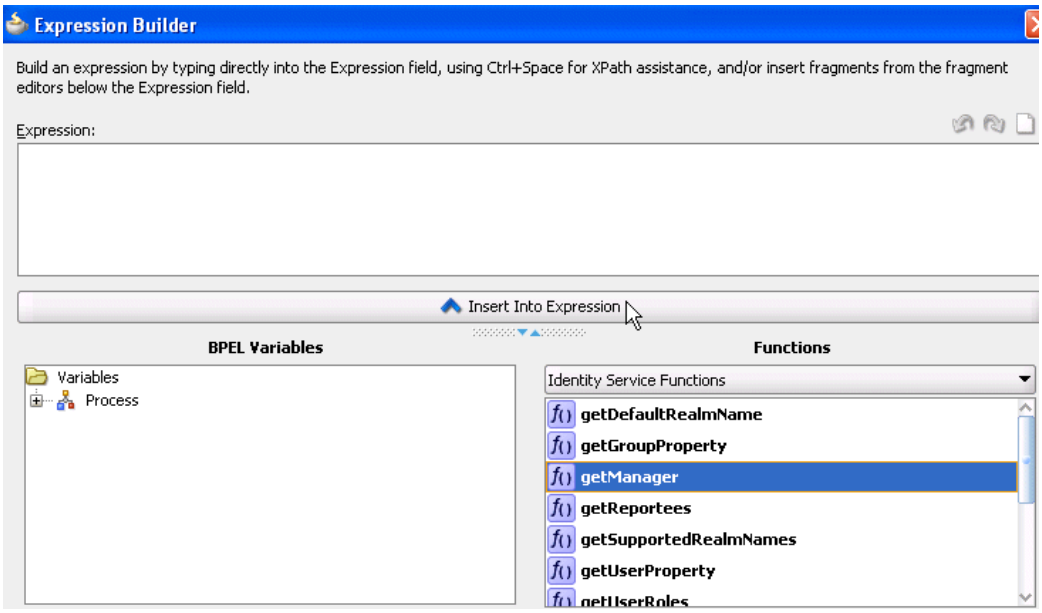
You can use the Expression Builder dialog and the XPath Building Assistant to create XPath expressions. You can visually design XPath expressions in a BPEL process, human workflow, or Oracle Mediator service component in the Expression Builder dialog.

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.

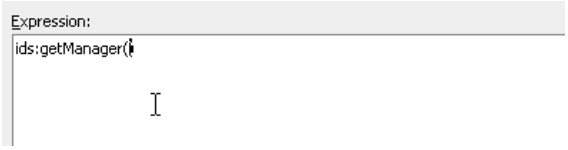
Figure B-1 Expression Builder Dialog



This inserts the function into the **Expression** field at the top.

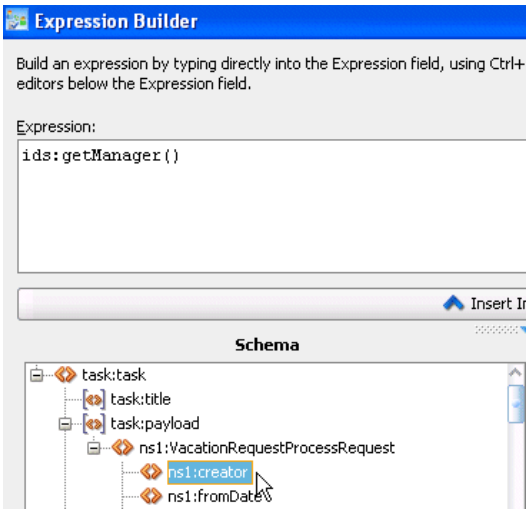
4. In the **Expression** field, place the cursor between the parentheses of the function, as shown in Figure B-2.

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.

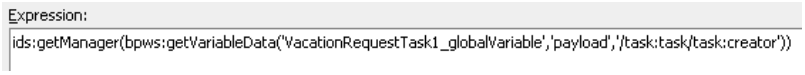
Figure B-3 Selection of Schema



6. Click **Insert Into Expression**.

   The expression is inserted into the function, as shown in Figure B-4.

Figure B-4 XPath Expression Creation



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 Map Editor

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

How to Use the XPath Building Assistant

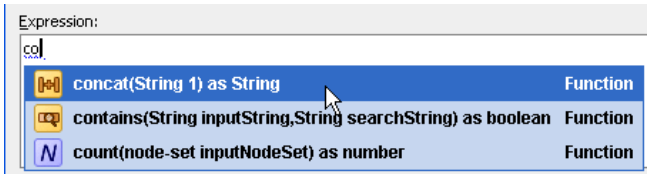
This section provides an example of using the XPath Building Assistant to build an expression in the **Expression** field of the Expression Builder dialog.

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.

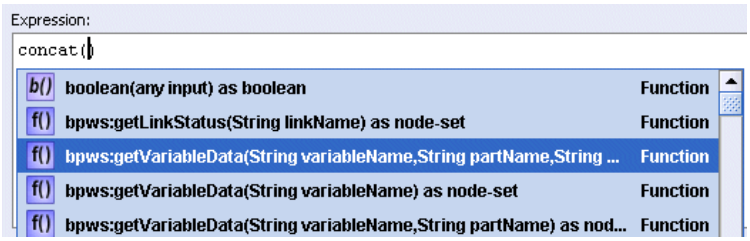
   Figure B-5 provides details.

Figure B-5 List of Values for Building an Expression



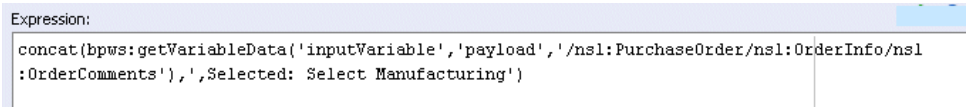
   This value is added to the **Expression** field. The list automatically displays again with different options and prompts you to enter the next portion of the XPath expression.
3. Select and double-click the next portion. Figure B-6 provides details.

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.

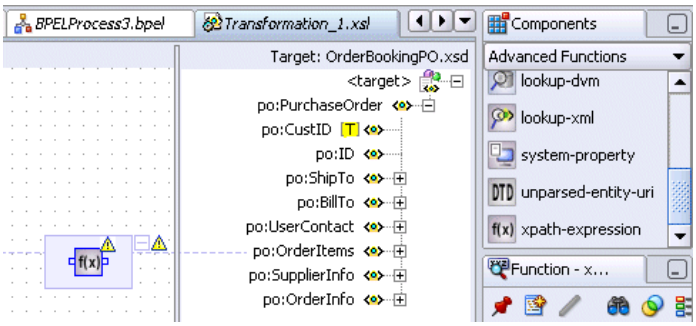
Using the XPath Building Assistant in the XSLT Mapper

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:

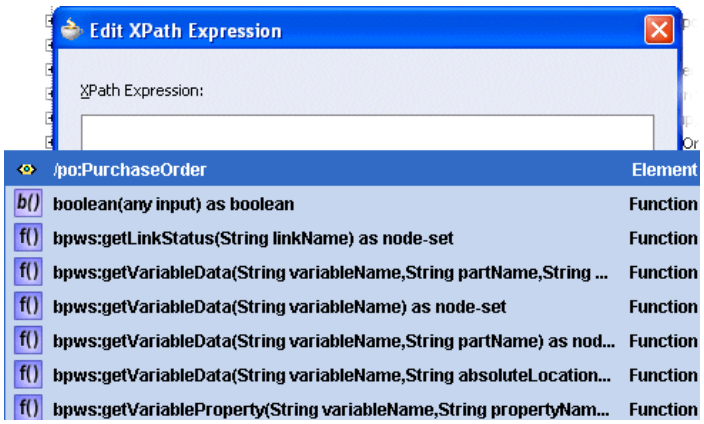
1. Go to the XSLT Map Editor.
2. From the **Component Palette** list, select **Advanced Functions**.
3. Scroll down the list to the **xpath-expression** function.
4. Drag and drop the **xpath-expression** function into the XSLT Map Editor, as shown in Figure B-8.

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

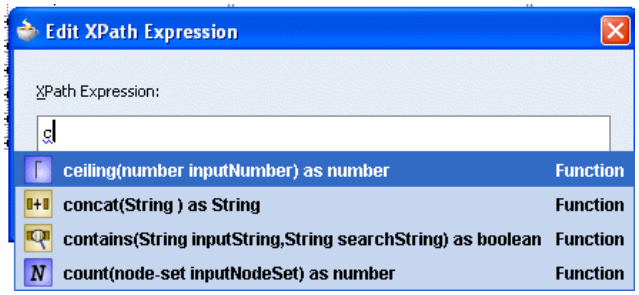


8. Make a selection from the list (for this example, **concat(String) as String**) in either of the following ways:

   - 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.

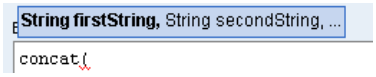
9. Continue this process to build the remaining parts of the XPath expression.

10. Click **OK** to close the Edit XPath Expression dialog when complete.

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

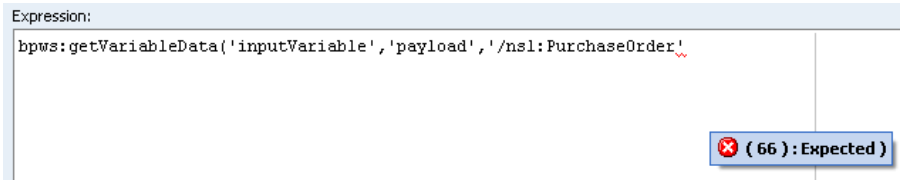


Once you finish specifying one argument, and enter a comma to move to the next argument, the tool tip updates itself to highlight the second argument name in bold, and so on. While editing existing XPaths that contain functions, you can re-invoke parameter tool tips by positioning the cursor within the function and then pressing a combination of the Ctrl, Shift, and space bar keys.

Syntactic and Semantic Validation

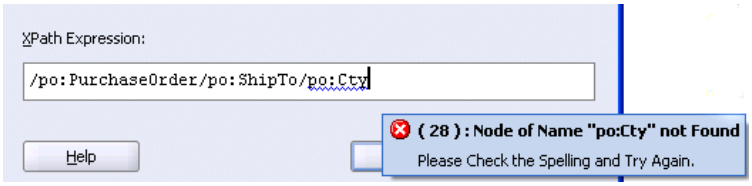
Within Oracle JDeveloper, an XPath expression is considered syntactically valid if it conforms to the XPath 1.0 specification. The XPath Building Assistant warns you about syntactically incorrect XPath functions (for example, a missing parenthesis or apostrophe) by underlining the erroneous area in red. 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 XPath functions 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

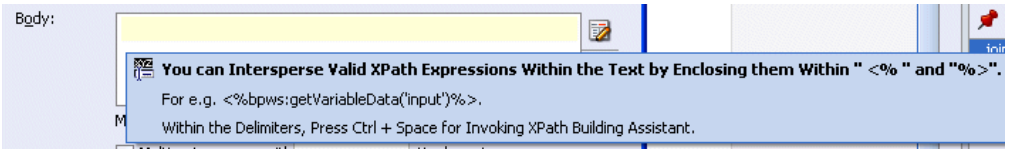


Creating Expressions with Free Form Text and XPath Expressions

You can mix free form text with XPath expressions in some dialogs.

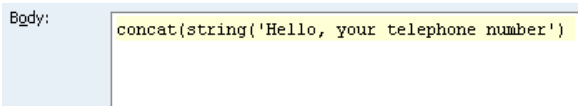
1. Place your cursor over the field to display a popup message that describes this functionality. Figure B-14 provides details.

Figure B-14 Functionality Description Menu



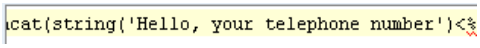
2. Enter free form text (in this example, `'Hello, your telephone number'`). Figure B-15 provides details.

Figure B-15 Free Form Text



3. Enter <% when you are ready to invoke the XPath Building Assistant. Figure B-16 provides details.

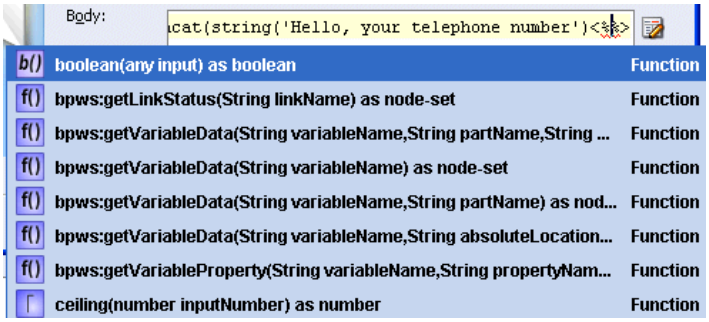
Figure B-16 XPath Building Assistant Invocation Preparation



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.

Using Double Slashes for Directory Paths in XPath Functions on Windows Can Cause Errors

The use of slashes to represent directory paths in XPath extension functions on Windows operating systems can be interpreted in two ways:

- With double slashes. For example, `file://c:/Ftab.txt`.
- With single slashes. For example, `file:/c:/Ftab.txt`.

If you specify double slashes and receive an error message, try specifying single slashes.

For example, the following use of double slashes does not work:

```
oraext:get-content-from-file-function("file://c:/Ftab.txt","file:
//c:/Ftab_1.xsd","root")
```

Whereas, the following use of single slashes works correctly:

```
oraext:get-content-from-file-function("file:/c:/Ftab.txt","file:
/c:/Ftab_1.xsd","root")
```

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`. You must implement XSLT Mapper functions differently than Oracle BPEL Process Manager, Oracle Mediator, and human workflow functions.

  For more information about these implementation differences, see 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`.

The following example shows the function schema used by system and user-defined functions:

```
<?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>
```

How to Implement User-Defined XPath Extension Functions

This section describes how to implement user-defined XPath extension functions for Oracle SOA Suite components.

How to Implement Functions for the XSLT Mapper

Implementation of user-defined XPath extension functions for the XSLT Map Editor is different than for other components:

- Each XSLT Map Editor function requires a corresponding public static method from a public static class. The function name and method name must match.
- XSLT Map Editor 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.

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 the following example:

```
package oracle.fabric.common.xml.xpath;
public interface IXPathFunction
{
   /** Call this function.
   *
   *  @param context The context at the point in the
   *         expression when the function is called.
   *  @param args List of arguments provided during
   *         the call of the function.
   */
   public Object call(IXPathContext context, List args) throws
XPathFunctionException;
}
```

   where:

   - `context`: The context at the point in the expression when the function is called.
   - `args`: The list of arguments provided during the call of the function.

   For the following example, a function named `getNodeValue(`*`arg1`*`)` is implemented that gets a value of `w3c node`:

```
package com.collaxa.cube.xml.xpath.dom.functions;
 import oracle.fabric.common.xml.xpath.IXPathFunction;
 import oracle.fabric.common.xml.xpath.IXPathFunction
 . . .

 public class GetNodeValue implements IXPathFunction {
    Object call(IXPathContext context, List args) throws XPathFunctionException
 {
        org.w3c.dom.Node node = (org.w3c.dom.Node) args.get(0);
        return node.getNodeValue()
    }
 }
```

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. The following example shows a sample configuration file that follows the function

schema shown in Creating User-Defined XPath Extension Functions. In this example, two functions are created: `mf:myFunction1` and `mf:myFunction2`.

```
<?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-3 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 national language support (NLS) for all function configurations.

Table B-3 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.

Table B-3 (Cont.) Function Schema Elements

Element	Description
`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.
`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-4 describes the naming conventions to use for user-defined configuration files.

Table B-4 User-Defined Configuration Files

To Use with This Component...	Use This Configuration File Name...
Oracle BPEL Process Manager	`ext-bpel-xpath-functions-config.xml`

Table B-4 (Cont.) User-Defined Configuration Files

To Use with This Component...	Use This Configuration File Name...
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.

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 Adding Custom Classes and JAR Files.

C

Deployment Descriptor Properties

This appendix describes how to define deployment descriptor configuration and partner link properties for BPEL process service components used at runtime by Oracle WebLogic Server, Oracle Enterprise Manager Fusion Middleware Control, or both.

This appendix includes the following section:

- Introduction to Deployment Descriptor Properties

For more information about deployment descriptor properties, see Chapter "Oracle BPEL Process Manager Performance Tuning" of *Tuning Performance*.

Introduction to Deployment Descriptor Properties

Deployment descriptors are BPEL process service component properties used at runtime by Oracle WebLogic Server, Oracle Enterprise Manager Fusion Middleware Control, or both. There are two types of properties:

- Configuration
- Partner link binding

Table C-1 lists the configuration deployment descriptor properties.

When you define configuration properties, you must add a prefix of `bpel.config` to the property name. For example, the property `inMemoryOptimization` must be defined as `bpel.config.inMemoryOptimization`. For information on defining properties in the Property Inspector in Oracle JDeveloper, see How to Define Deployment Descriptor Properties in the Property Inspector.

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. **Note**: When an unhandled fault occurs, regardless of these flags, audit information for instances is persisted within the `CUBE_INSTANCE` table. • `off`: No instances of this process are saved.
`disableAsserts`	This property, when set to `true`, disables assertions in BPEL projects.

Table C-1 (Cont.) Properties for the configurations Deployment Descriptors

Property Name	Description
`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 are persisted in the database. With this setting, reliability is obtained with some performance impact on the database. In some cases, overall system performance can be impacted. • `async.cache`: Incoming delivery messages are kept only in the in-memory cache. If performance is preferred over reliability, consider this setting. When set to `async.cache`, if the rate at which one-way messages arrive is much higher than the rate at which they are delivered, or if the server fails, messages can be lost. In addition, the system can become overloaded (messages become backlogged in the scheduled queue) and you can receive out-of-memory errors. Consult your own use case scenarios to determine if this setting is appropriate. When you set `oneWayDeliveryPolicy` to `async.cache` in high availability environments, invoke and callback messages in the middle of execution at the time of a server crash may be lost or duplicated. Server failover is not supported for `async.cache`. For more information, see Section "Oracle BPEL Process Manager High Availability Architecture and Failover Considerations" of *High Availability Guide*. • `sync`: Direct invocation occurs on the same thread. The scheduling of messages in the invoke queue is bypassed, and the BPEL instance is invoked synchronously. In some cases this setting can improve database performance. For information about setting this property during BPEL process creation, see How to Add a BPEL Process Service Component. For more information, see Section "Tuning Database Persistence for BPEL" of *Tuning Performance*.
`reenableAggregationOnComplete`	This property controls the number of instances to create and use to route messages. The possible values are: • `true`: Creates a new instance to handle the messages of the same correlation. • `false`: Creates only one instance for handling messages. For more information, see Routing Messages to the Same Instance.

Table C-1 (Cont.) Properties for the configurations Deployment Descriptors

Property Name	Description
`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. • `notSupported`: Executes a business process without the need for a transaction. For more information, see Executing a Business Process Without a 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 a correlation is needed and it is always done asynchronously. For information about setting this property during BPEL process creation, see How to Add a BPEL Process Service Component.

Table C-2 lists the partner link binding deployment descriptor properties.

When you define partner link binding properties, you must add a prefix of `bpel.partnerLink.`*`partner_link_name`* to the property name. For example, the property `nonBlockingInvoke` must be defined as `bpel.partnerLink.`*`partner_link_name`*`.nonBlockingInvoke`. For information on defining properties in the Property Inspector in Oracle JDeveloper, see How to Define Deployment Descriptor Properties in the Property Inspector.

Table C-2 Properties for the partnerLinkBinding Deployment Descriptors

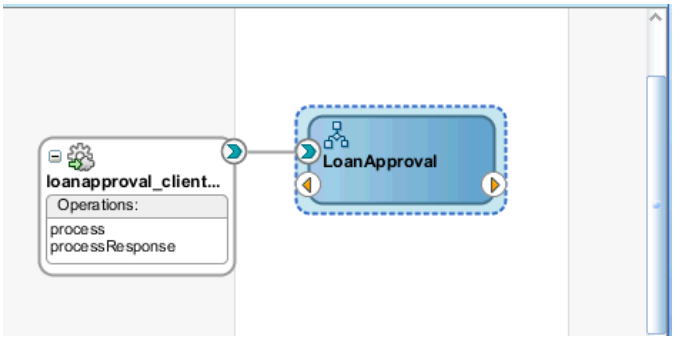
Property Name	Description
`idempotent`	An idempotent activity is an activity that can be retried (for example, an assign activity or an invoke activity). The instance is saved after a nonidempotent activity. This property is applicable to both durable and transient processes. • `true` (default): If the 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. For information about using fault handling with the `idempotent` property set to `false`, see What You May Need to Know About the idempotent Property and Fault Handling. For more information about the `idempotent` property, see Managing Idempotence at the Partner Link Operation Level.
`nonBlockingInvoke`	Default value is `false`. When this is set to `true`, a separate thread is spawned to perform the invocation so that the invoke activity does not block the instance. For more information, see What You May Need to Know About the Execution of Parallel Flow Branches in a Single Thread.
`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.

How to Define Deployment Descriptor Properties in the Property Inspector

You define configuration and partner link binding deployment descriptor properties and values in the Property Inspector of Oracle JDeveloper. When complete, the properties are displayed in the BPEL process service component section of the `composite.xml` file.

1. In the SOA Composite Editor, select the BPEL process service component, as shown in Figure C-1.

Figure C-1 Selected BPEL Process Service Component

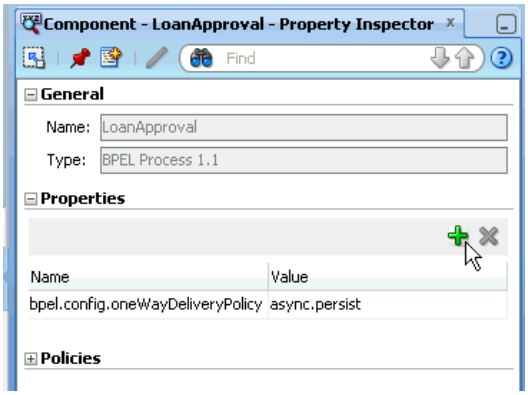


2. Go to the Property Inspector in the lower right corner of Oracle JDeveloper.

3. In the **Properties** section, click the **Add** icon, as shown in Figure C-2.

   For this example, the **oneWayDeliveryPolicy** property is already defined because the **Delivery** option was selected in the Create BPEL Process dialog during BPEL process creation. For more information about setting this property during BPEL process creation, see How to Add a BPEL Process Service Component.

   Figure C-2 Property Inspector

   The Create Property dialog is displayed.

4. In the **Name** field, enter the deployment descriptor property. For this example, the configuration deployment descriptor property `oneWayDeliveryPolicy` is defined. Therefore, a prefix of `bpel.config` is required. For more information about configuration deployment descriptor properties, see Table C-1.

   If you instead add a partner link binding property, a prefix of `bpel.partnerLink.`*`partner_link_name`* is required, where *`partner_link_name`* is the name of the partner link (for example, `LoanService`). For more information about partner link binding deployment descriptor properties, see Table C-2.

5. In the **Value** field, enter an applicable value for this property (for example, `async.persist`).

6. Click **OK**.

   The Property Inspector displays the added deployment descriptor property.

7. Click **Source** in the SOA Composite Editor.

   The `oneWayDeliveryPolicy` configuration property with the `bpel.config` prefix is displayed in the `composite.xml` file, as shown in the following example:

```
<component name="LoanApproval" version="2.0">
. . .
  <componentType>
  . . .
  . . .
  </componentType>
  <property name="bpel.config.oneWayDeliveryPolicy" type="xs:string"
  many="false">async.persist</property>
  </component>
```

If you instead define a partner link binding deployment descriptor property in the Property Inspector (for example, the `nonBlockingInvoke` partner link binding

property), it is displayed in the `composite.xml` file, as shown in the example that follows. Note the prefix of `bpel.partnerLink.`*`partner_link_name`*, which is required for this type of property.

```
<component name="myBPELServiceComponent" version="2.0">
 . . .
  <componentType>
  . . .
  . . .
  </componentType>
<property name="bpel.partnerLink.partner_link_name.nonBlockingInvoke">
false</property>
</component>
```

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.

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:

- Introduction to Sensor Public Views and the Sensor Actions XSD File
- Sensor Public Views
- Sensor Actions XSD File

For more information, see Using Sensors and Analytics .

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.

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.

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.

BPEL_PROCESS_INSTANCES

Table D-1 provides an overview of all the process instances of .

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

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
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

Table D-2 (Cont.) BPEL_ACTIVITY_SENSOR_VALUES View

Attribute Name	SQL Type	Attribute Size	Indexed or Unique?	Null?	Comment
COMPONENT_NAME	VARCHAR2	500	--	N	User-defined component name

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
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

Table D-3 (Cont.) BPEL_FAULT_SENSOR_VALUES View

Attribute Name	SQL Type	Attribute Size	Indexed or Unique?	Null?	Comment
ACTIVITY_TYPE	VARCHAR2	30	--	N	The type of the BPEL activity
MESSAGE	CLOB	--	--	Y	The fault message

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
TS_DATE	DATE	--	--	N	Date portion of `creation_date`
TS_HOUR	NUMBER	--	--	N	Hour portion of `creation_date`
VARIABLE_NAME	NVARCHAR2	512	--	N	The name of the BPEL variable

Table D-4 (Cont.) BPEL_VARIABLE_SENSOR_VALUES View

Attribute Name	SQL Type	Attribute Size	Indexed or Unique?	Null?	Comment
EVAL_POINT	VARCHAR2	30	--	Y	Evaluation point of the corresponding activity sensor
CRITERIA_SATISFIED	VARCHAR2	1	--	Y	NULL, Y, or N
TARGET	NVARCHAR2	512	--	--	--
UPDATER_NAME	NVARCHAR2	200	--	N	The name of the activity or event that updated the variable
UPDATER_TYPE	NVARCHAR2	200	--	N	The type of the BPEL activity or event
SCHEMA_NAMESPACE	NVARCHAR2	512	--	Y	Namespace of variable sensor value
SCHEMA_DATATYPE	NVARCHAR2	512	--	Y	Data type of the variable sensor value
VALUE_TYPE	NUMBER	--	--	N	The value type of the variable (corresponds to java.sql.Types values)
VARCHAR2_VALUE	NVARCHAR2	4000	--	Y	The value of string-like variables
NUMBER_VALUE	NUMBER	--	--	Y	
DATE_VALUE	TIMESTAMP	6	--	Y	User-defined date
DATE_VALUE_TZ	VARCHAR2	10	--	Y	User-defined time zone
BLOB_VALUE	BLOB	--	--	Y	
CLOB_VALUE	CLOB	--	--	Y	

Sensor Actions XSD File

The following example provides a sample sensor action schema that you can import into Oracle BPEL Designer. This schema is also relevant to custom data publishers.

```
<?xml version="1.0" encoding="utf-8"?>
<!--
  This schema contains the sensor definition. Sensors monitor data
  and execute callbacks appropriately.

  BPEL designer uses this file as a template to generate to generate
  SensorActionData.xsd. It does this by replacing special tags.
  Do not modify these special tags. For details, see comments in the file.
  The replacement is done using a simple text replacement, so the white
  spaces too should be preserved as indicated in comments.
-->
<xsd:schema blockDefault="#all" elementFormDefault="qualified"
            targetNamespace="http://xmlns.oracle.com/bpel/sensor"
```

```
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:sensor="http://xmlns.oracle.com/bpel/sensorDataPlaceHolder"
            xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd"
            xmlns:tns="http://xmlns.oracle.com/bpel/sensor"
            nxsd:encoding="UTF-8">

<!-- *** The following line is a place holder. Do not remove it. It must remain as
 is, including any whitespace.  If you change this, please let BAM sensor action
 developer know. -->
<!-- $importSensorVar -->

  <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="BAM"/>
      <xsd:enumeration value="LogFile"/>
      <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:enumeration value="service"/>
      <xsd:enumeration value="reference"/>
      <xsd:enumeration value="event"/>
    </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="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="tSensorConfig"/>

<xsd:complexType name="tExpressionConfig">
  <xsd:complexContent>
    <xsd:extension base="tns:tVariableConfig">
      <xsd:attribute name="expression" use="required" type="xsd:string">
        <xsd:annotation>
          <xsd:documentation>
             expresion="$in/$payload/$partName/xpathExpression |
                        $in/$header/xpathExpression |
                        $in/$property/name |
                        $out/$payload/$partName/xpathExpression |
                        $out/$header/xpathExpression |
                        $out/$property/name |
                        $fault/$payload/$partName/xpathExpression |
                        $fault/$header/xpathExpression |
                        $fault/$property/name"

               Where
                  $in - The input/request message to the operation/event
                  $out - The output/Response message from the operation
                  $fault - The fault message from the operation
           </xsd:documentation>
        </xsd:annotation>
      </xsd:attribute>          </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tOperationConfig">
  <xsd:complexContent>
    <xsd:extension base="tns:tExpressionConfig">
      <xsd:attribute name="operation" use="required" type="xsd:string">
        <xsd:annotation>
          <xsd:documentation>
             The name of the operation in the service/reference on which the
sensor is defined.
          </xsd:documentation>
        </xsd:annotation>
      </xsd:attribute>
    </xsd:extension>
  </xsd:complexContent>
```

```
  </xsd:complexType>

  <xsd:complexType name="tServiceConfig">
    <xsd:complexContent>
      <xsd:extension base="tns:tOperationConfig">
        <xsd:attribute name="service" use="required" type="xsd:string">
          <xsd:annotation>
            <xsd:documentation>
               The name of the service on which the sensor is defined.
            </xsd:documentation>
          </xsd:annotation>
        </xsd:attribute>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tReferenceConfig">
    <xsd:complexContent>
      <xsd:extension base="tns:tOperationConfig">
        <xsd:attribute name="reference" use="required" type="xsd:string">
          <xsd:annotation>
            <xsd:documentation>
               The name of the reference on which the sensor is defined.
            </xsd:documentation>
          </xsd:annotation>
        </xsd:attribute>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tEventConfig">
    <xsd:complexContent>
      <xsd:extension base="tns:tExpressionConfig">              <xsd:attribute
name="component" type="xsd:string" use="required">
          <xsd:annotation>
            <xsd:documentation>
               The name of the component which raises or receives the event.
            </xsd:documentation>
          </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="event" use="required" type="xsd:string">
          <xsd:annotation>
            <xsd:documentation>
               The name of the event that the component raises or receives.
            </xsd:documentation>
          </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="actionType" use="required">
          <xsd:simpleType>
            <xsd:restriction base="xsd:string">
              <xsd:enumeration value="Publish"/>
              <xsd:enumeration value="Subscribe"/>                    </
xsd:restriction>
          </xsd:simpleType>
        </xsd:attribute>            </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tSensor">
    <xsd:sequence>
      <xsd:element name="activityConfig" type="tns:tActivityConfig"
```

```
minOccurs="0"/>
     <xsd:element name="faultConfig" type="tns:tFaultConfig" minOccurs="0"/>
     <xsd:element name="variableConfig" type="tns:tVariableConfig"
minOccurs="0"/>
     <xsd:element name="serviceConfig" type="tns:tServiceConfig" minOccurs="0"/>
     <xsd:element name="referenceConfig" type="tns:tReferenceConfig"
minOccurs="0"/>
     <xsd:element name="eventConfig" type="tns:tEventConfig" 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="target" use="required" type="xsd:string"/>
   <xsd:attribute name="filter" 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="tProperty">
   <xsd:simpleContent>
     <xsd:extension base="xsd:string">
       <xsd:attribute name="name" use="required" type="xsd:string"/>
     </xsd:extension>
   </xsd:simpleContent>
 </xsd:complexType>

 <xsd:complexType name="tHeaderInfo">
   <xsd:sequence>
     <xsd:element name="applicationName" type="xsd:string"/>
     <xsd:element name="compositeName" type="xsd:string"/>
     <xsd:element name="compositeInstanceId" type="xsd:string"/>
     <xsd:element name="compositeRevision" type="xsd:string"/>
     <xsd:element name="compositeLabel" type="xsd:string"/>
     <xsd:element name="componentName" type="xsd:string"/>
     <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:element name="property" minOccurs="0" maxOccurs="unbounded"
type="tns:tProperty"/>
   </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="variableData" type="tns:tVariableData" minOccurs="0"
maxOccurs="unbounded"/>
       <xsd:element name="serviceData" type="tns:tServiceData" minOccurs="0"
maxOccurs="unbounded"/>
       <xsd:element name="referenceData" type="tns:tReferenceData" minOccurs="0"
maxOccurs="unbounded"/>
       <xsd:element name="eventData" type="tns:tEventData" minOccurs="0"
```

```
maxOccurs="unbounded"/>
     </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="faultName" type="xsd:QName"/>
     <!-- *** The following line is a place holder. Do not remove it. It must
remain as is, including any whitespace.  If you change this, please let BAM
sensor action developer know. -->
     <xsd:element name="data" type="xsd:anyType" minOccurs="0"/> <!-- DO NOT
MODIFY: fault data type -->
   </xsd:sequence>
 </xsd:complexType>


 <!--
  xml type that will be provided to sensors for variable Datas. Note the
     any element represents variable data.
  -->
 <xsd:complexType name="tVariableData">
   <xsd:sequence>
     <xsd:element name="dataType" type="xsd:integer"/>
     <!-- *** The following line is a place holder. Do not remove it. It must
remain as is, including any whitespace.  If you change this, please let BAM
sensor action developer know. -->
     <xsd:element name="data" type="xsd:anyType"/> <!-- DO NOT MODIFY: sensor
variable data type -->
     <xsd:element name="queryName" type="xsd:string"/>
     <xsd:element name="target" type="xsd:string"/>
     <xsd:element name="updaterName" type="xsd:string" minOccurs="1"/>
     <xsd:element name="updaterType" type="xsd:string" minOccurs="1"/>
   </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="tServiceData">
   <xsd:sequence>
     <xsd:element name="sensorName" type="xsd:string"/>
     <xsd:element name="data" type="xsd:anyType"/>
     <xsd:element name="dataType" type="xsd:integer"/>
   </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="tReferenceData">
   <xsd:sequence>
     <xsd:element name="sensorName" type="xsd:string"/>
     <xsd:element name="data" type="xsd:anyType"/>
     <xsd:element name="dataType" type="xsd:integer"/>
   </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="tEventData">
   <xsd:sequence>
     <xsd:element name="sensorName" type="xsd:string"/>
     <xsd:element name="data" type="xsd:anyType"/>
     <xsd:element name="dataType" type="xsd:integer"/>
   </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="durationInSeconds" minOccurs="0" type="xsd:double"/>
     <xsd:element name="duration" type="xsd:duration" minOccurs="0"/>
     <xsd:element name="errorMessage" nillable="true" minOccurs="0"
type="xsd:string"/>
   </xsd:sequence>
 </xsd:complexType>

 <!--
   The header of the document contains some metadata.
 -->
              <!--
   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>

-->

    <!--
    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>

<!--
  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="variableData" type="tns:tProcessVariableData"
minOccurs="0" maxOccurs="unbounded"/>
  </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

Propagating Normalized Message Properties Through Message Headers

This appendix describes how to set normalized message properties that enable you to propagate these properties through message headers.

This appendix includes the following sections:

- Introduction to Normalized Messages
- Manipulating Normalized Message Properties with bpelx Extensions

Introduction to Normalized Messages

Header manipulation and propagation is a key business integration messaging requirement. Components such as Oracle BPEL Process Manager, Oracle Mediator, Oracle JCA adapters, REST 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.

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 information about normalized message properties in JCA adapters and Oracle B2B, see *Understanding Technology Adapters* and *User's Guide for Oracle B2B*.

Oracle Web Services Addressing Properties

Table E-1 lists the predetermined properties of a normalized message for Web Services Addressing (WS-Addressing). The WS-Addressing headers from incoming SOAP

requests are propagated *within* Oracle SOA Suite through the normalized message properties. However, overriding of WS-Addressing headers in the outbound SOAP message through use of these normalized message properties is not supported.

Table E-1 Properties for Oracle Web Services Addressing

Property Name	Propagatable (Yes/No)	Direction (Inbound / Outbound)	Data Type	Range of Valid Values	Description
`wsa.messageId`	No	Inbound	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	Inbound	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	Inbound	String	URI format	Represents a contract between two components communicating asynchronously.
`wsa.replyToPortType`	No	Inbound	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	Inbound	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.

Table E-1 (Cont.) Properties for Oracle Web Services Addressing

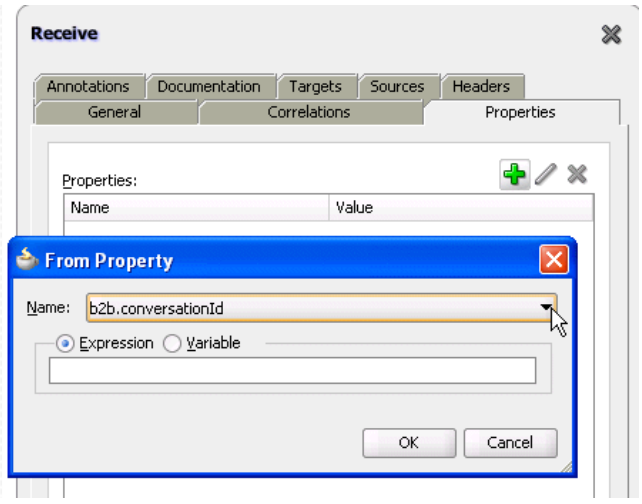
Property Name	Propagatable (Yes/No)	Direction (Inbound / Outbound)	Data Type	Range of Valid Values	Description
`wsa.action`	No	Inbound	String	URI format	This required element (whose content is of type `xs:anyURI`) conveys the value of the action property.
`wsa.to`	No	Inbound	String	URI format	This optional 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`.

How to Set Normalized Message Properties in Message Headers

To set normalized message properties in message headers:

1. In the dialog of the selected activity, click the **Properties** tab.
2. For BPEL 2.0 projects, perform the following tasks:
   a. Click the **Add** icon.
   b. From the **Name** list, select the property. Figure E-1 provides details.

Figure E-1 Properties Tab for Normalized Messages Header Properties



c. Select the value of the property:

If You Select...	Perform the Following Steps...
Expression	**i.** Click **Search** to invoke the XPath Expression Builder dialog. **ii.** Create the XPath expression, and click **OK**. **iii.** Click **OK**.
Variable	**i.** Click **Search** to invoke the Variable XPath Builder dialog. **ii.** Select the variable, and click **OK**. **iii.** Click **OK**.

The defined property is displayed.

3. For BPEL 1.1 projects, perform the following tasks:

   a. Scroll down and select the property.

   b. In the **Value** column, double-click to display the ellipses.

   c. Click the ellipses.

   The Adapter Property Value dialog is displayed.

   d. Enter the variable name as the value, and click **OK**.

   e. For activities with a **Type** column (for example, invoke activities), click the row of the property.

   f. From the list that is displayed, select **input** or **output** for the message direction.

   g. Click **Apply**, then **OK**.

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.

BPEL 2.0 bpelx Extensions Syntax

The following example shows `bpelx` extensions syntax 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`.

BPEL 1.1 bpelx Extensions Syntax

The following example shows `bpelx` extensions syntax 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>
```

F

Interfaces Implemented By Rules Dictionary Editor Task Flow

This appendix describes the Oracle Business Rules Dictionary Editor Task Flow, which implements the `MetadataDetails` and `NLSPrefrences` interfaces when creating an ADF-based Web application. The interfaces are defined in the `soaComposerTemplates.jar` file.

This appendix includes the following sections:

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 below:

```
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;
}
```

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.

The code sample below shows how to get the file from a local file system:

```
private static final String RULES_FILE1 =
"file:///C:/scratch/<username>/system/mywork/linkedD/AutoAppProj/oracle/rules/credit/
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();
    }
```

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. The code sample below shows how to find the path of the linked dictionaries that are stored within the `../oracle/rules` directory in a local file system:

```
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 "";
    }
```

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. The code sample below shows how to save the document in the local file system:

```
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);
                }
            }
        }
    }
```

The NLSPreferences Interface

The `NLSPrefrences` interface defines four methods as shown below:

```
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();

    /**
     * Returns the grouping seperator.
     */
    char getGroupingSeparator();

    /**
     * Returns the grouping seperator.
     */
   char getDecimalSeparator();
}
```

The code sample below shows a 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";
    private static final char G_SEP = ',';
    private static final char D_SEP = '.';

        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;
        }

        public char getGroupingSeparator() {
        return G_SEP;
    }
```

```
public char getDecimalSeparator() {
    return D_SEP;
    }
}
```

G

Oracle SOA Suite Configuration Properties Road Map

This appendix describes the locations 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:

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 Fusion Middleware Control, or both. You set these properties during design time in the `composite.xml` file of the SOA composite application. Examples of deployment descriptor properties include `completionPersistPolicy`, `inMemoryOptimization`, `oneWayDeliveryPolicy`, `transaction`, `nonBlockingInvoke`, and others.

For more information about available deployment descriptor properties, see How to Define Deployment Descriptor Properties in the Property Inspector and Transaction and Fault Propagation Semantics in BPEL Processes.

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
- REST adapters

For more information, see Propagating Normalized Message Properties Through Message Headers.

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 *Understanding Technology Adapters* for JCA adapter properties
- *Oracle E-Business Suite Adapter User's Guide* for Oracle Applications adapter properties

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 Propagating Normalized Message Properties Through Message Headers.

Oracle B2B Message Header Properties

In Oracle B2B, you can manipulate headers with reserved key words.

For more information about available Oracle B2B message header properties, see Appendix, "Back-End Applications Interface" of *User's Guide for Oracle B2B*.

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 G-1 lists these properties and provides references to documentation that describes how to configure these properties.

Table G-1 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.	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.	Handling Faults with 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.	Handling Faults with 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.	How to Assign Values How to Access Headers for Filters and Assignments
`rolesAllowed`	Specifies role names required to invoke SOA composite applications from any Java EE application.	Specifying Enterprise JavaBeans Roles

`streamIncomingAttachments`
and
`streamOutgoingAttachments`
	Specify these properties to stream attachments with SOAP.	SOAP with Attachments

`oracle.webservices.local.optimization`	Specifies to override a local optimization setting for a policy.	SOAP with Attachments and *Administering Oracle SOA Suite and Oracle Business Process Management Suite*
`oracle.soa.local.optimization.force`	You can override the `oracle.webservices.local.optimization` property and force optimization.	*Administering Oracle SOA Suite and Oracle Business Process Management Suite*
`one.way.returns.fault`	Controls how faults and one-way messages are handled for one-way interface SOAP calls.	One-way Message Exchange Patterns

Table G-1 (Cont.) Oracle SOA Suite Properties

Property	Description	See...
`mtomThreshold`	Specifies the attachment size in bytes.	Sending and Receiving MTOM-Optimized Messages to SOA Composite Applications

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 enter fault policy properties automatically through the Fault Policy Editor or manually in a fault policy framework file. Table G-2 lists these properties and provides references to documentation that describes how to configure these properties.

Table G-2 Oracle SOA Suite Fault Policy Properties

Property	Description	See...
`retryInterval`	Provides a delay between retries of an activity (in seconds).	Manually Creating a Fault Policy File for Automated Fault Recovery
`retryCount`	Retries an activity a specified number of times.	How to Design a Fault Policy for Automated Fault Recovery with the Fault Policy Wizard or Manually 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.	How to Design a Fault Policy for Automated Fault Recovery with the Fault Policy Wizard or 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 *Understanding Technology Adapters*.

Oracle B2B System Properties

You can set most Oracle 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 "Configuring B2B System Parameters" of *User's Guide for Oracle B2B*.

Oracle Healthcare Properties

You can configure Oracle Healthcare runtime and user interface, workflow notification, and normalized message header properties.

For more information about available Oracle Healthcare properties, see *Healthcare Integration User's Guide for Oracle SOA Suite*.

Oracle Business Activity Monitoring Properties

You can configure Oracle Business Activity Monitoring (BAM) business view properties.

For more information about Oracle BAM properties, see *Monitoring Business Activity with Oracle BAM*.

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, REST adapters, and Oracle Service Registry)
- Global token variables and automatic database purging

SOA Infrastructure Properties

You can configure properties for the SOA Infrastructure on the SOA Infrastructure Common Properties page. 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
- Payload validation
- Time duration during which to retrieve instances and faults data
- Universal Description, Discovery, and Integration (UDDI) registry
- Callback server and server URLs
- BPM Analytics, BPEL sensors, and composite sensors
- Java Naming and Directory Interface (JNDI) data source
- Web service binding properties
- Advanced configuration properties

For more information about available SOA Infrastructure properties, see Chapter "Configuring the SOA Infrastructure" of *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

Oracle BPEL Process Manager Properties

You can configure BPEL process service engine properties on the BPEL Service Engine Properties page. 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
- Payload schema validation
- BPEL monitor and sensor enabling
- Advanced configuration properties

For more information about available Oracle BPEL Process Manager properties, see Chapter "Configuring BPEL Process Service Components and Engines" of *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

Human Workflow Notification and Task Service Properties

You can configure human workflow notification and task service properties on the **Mailer** and **Task** tabs of the Workflow Notification Properties page. 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 email account name
- The workflow session time out and custom class path URL values
- The dynamic assignment and task escalation functions of the assignment service
- Advanced configuration properties

For more information about available human workflow notification and task service properties, see Chapter "Configuring Human Workflow Service Components and Engines" of *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

Oracle Mediator Properties

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 maximum rows retrieved
- Parallel locker thread sleep

- Custom configuration parameters
- Container ID refresh time and container ID lease timeout
- Resequencer locker thread sleep and maximum groups locked
- Advanced configuration properties

For more information about available Oracle Mediator properties, see Chapter "Configuring Oracle Mediator Service Components and Engines" of *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

Cross Reference Properties

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 "Managing Cross-References" of *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

Oracle B2B Properties

You can enable Oracle B2B Dynamic Monitoring Service (DMS) metrics and configure advanced properties.

For more information about available Oracle B2B properties, see Chapter "Configuring Oracle B2B" of *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

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 JCA adapters
- Web services properties such as enabling REST; enabling the WSDL, metadata exchange, and endpoint of the web service; and others

For more information about available service and reference binding component properties, see Chapter "Configuring Service and Reference Binding Components" of *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

Global Token Variables and Automatic Database Purging Properties

You can configure additional properties in Oracle Enterprise Manager Fusion Middleware Control:

- Define global token variables for specific URIs in SOA composite applications.
- Enable automatic purging of large numbers of instances from the database.

For more information about token configurations, see the "Managing Global Token Variables for Multiple SOA Composite Applications" section of *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

For more information, see the "Deleting Large Numbers of Instances with Oracle Enterprise Manager Fusion Middleware Control" section of *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

System MBean Browser Advanced 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 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 B2B

Note:

In addition to advanced properties, the same properties that display for modifying in the property pages described in Property Pages also display for modifying in the System MBean Browser.

SOA Infrastructure Advanced Properties

The **More SOA Infra Advanced Configuration Properties** link at the bottom of the SOA Infrastructure Common Properties page enables you to display System MBean Browser advanced properties for the SOA Infrastructure. Properties that display for modifying include, but are not limited to, 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

For more information about available SOA Infrastructure System MBean Browser properties, see Chapter "Configuring the SOA Infrastructure" of *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

Oracle BPEL Process Manager Advanced Properties

The **More BPEL Configuration Properties link** at the bottom of the BPEL Service Engine Properties page enables you to display System MBean Browser properties for the BPEL process. Properties that display for modifying include, but are not limited to, 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

For more information about available Oracle BPEL Process Manager System MBean Browser properties, see Chapter "Configuring BPEL Process Service Components and Engines" of *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

Oracle Mediator Advanced Properties

The **More Mediator Configuration Properties** link at the bottom of the Mediator Service Engine Properties page enables you 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 "Configuring Oracle Mediator Service Components and Engines" of *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

Human Workflow Notification and Task Service Advanced Properties

The **More Workflow Notification Configuration Properties** link at the bottom of the Workflow Notification Properties page and the **More Workflow Task Service Configuration Properties** link at the bottom of the Workflow Task Service Properties page enables you to display System MBean Browser properties for human workflow. Properties that display for modifying include, but are not limited to, the following:

- The address at which to receive incoming instant messages (IMs)
- Whether to return custom notification service property names

For more information about available human workflow notification and task service System MBean Browser properties, see Chapter "Configuring Human Workflow Service Components and Engines" of *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

Oracle B2B Advanced Properties

The **More B2B Configuration Properties** link at the bottom of the B2B Server Properties page enables you to display System MBean Browser properties for Oracle B2B. Properties that display for modifying include, but are not limited to, Oracle B2B payload obfuscation.

For more information about available Oracle B2B properties, see Chapter "Configuring Oracle B2B" of *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

H

Working with Large Schemas in the XSLT Editor

The XSLT Editor displays source and target trees that provide an XML representation of the input and output documents for the XSLT map that is being edited. The editor creates these trees from the XSD schema documents after you select a root element definition.

These schema trees can become large and difficult to work with in a graphical mapping tool, such as the XSLT Editor. Some schema documents define hundreds of child nodes for each parent node. Expanding a few parent nodes like this, in the tree, can generate thousands of tree nodes to scroll through when trying to create an XSLT Map.

> If the mapping is sparse, i.e. there are only a few mapped target nodes even though the schema is very large, the user needs to constantly scroll through nodes that do not need to be mapped. On the other hand, if the mapping is not sparse, and many mappings exist, the user faces a lot of crisscrossing lines that make it difficult to make sense out of the mappings.

This appendix discusses strategies for both sparse and non-sparse maps, as well as ways to reduce clutter.

Sparse Mappings

Schemas are often created to handle a large range of possibilities. When schemas of this type are used to produce source and target trees, the trees can contain hundreds of thousands, or even millions of nodes. However, in many cases, the user is only interested in using or populating a small portion of the nodes defined in the schema.

There are various ways of handling sparse mappings.

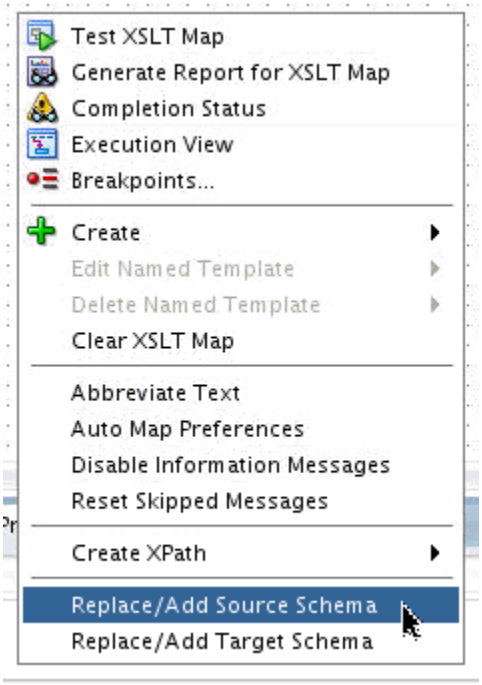
Using Sample XML to Generate a Schema

The 12*c* XSLT Editor has the ability to create schemas from XML documents that can then be used as schema documents for a source or target tree. If you have a sample XML document for your source and/or target, this document can be used to build small schema documents that contain only those nodes that you need for the map.

To create an XSLT map using sample source and target XML documents, select the **Generate from XML** option while selecting the schema for a source or target in a new XSL map.

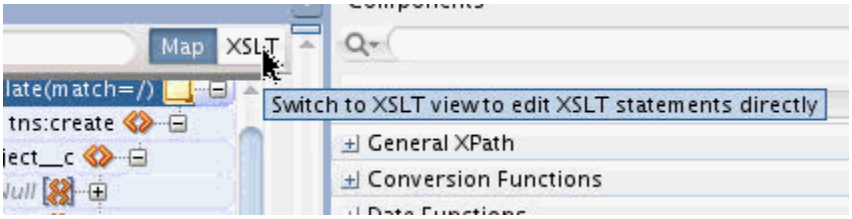
A schema is generated and placed in the `Schemas` folder. This schema will be used to create the source and target trees for your mapping, and consequently will contain only the nodes that exist in your original sample XML document.

If you wish to switch back and forth from the small sample schema to the larger schema that you might be avoiding, you can select **Replace/Add Source or Target Schema** from the canvas context menu. Then select either the small sample schema from the Schemas folder or the larger schema.

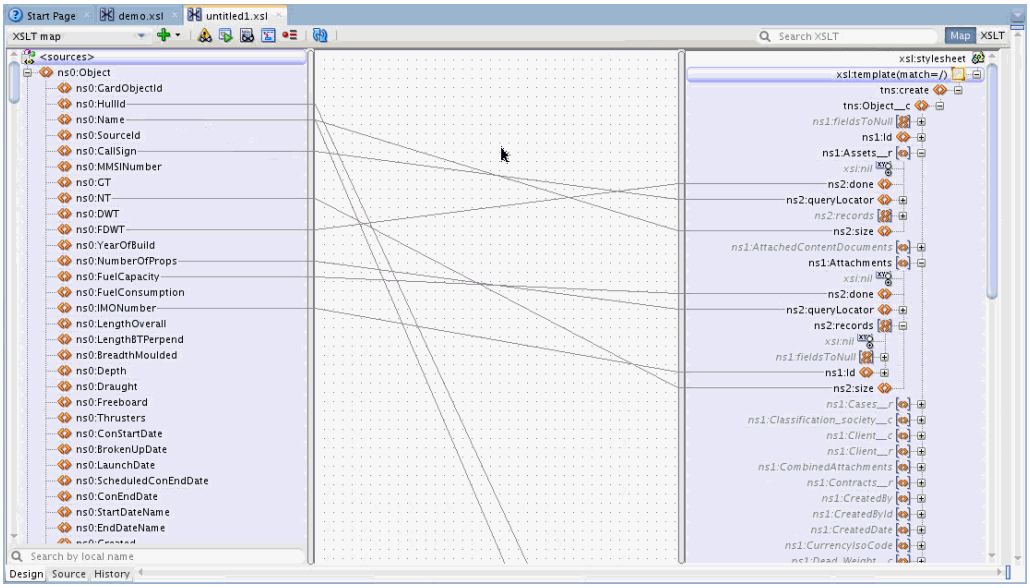


Using XSLT View

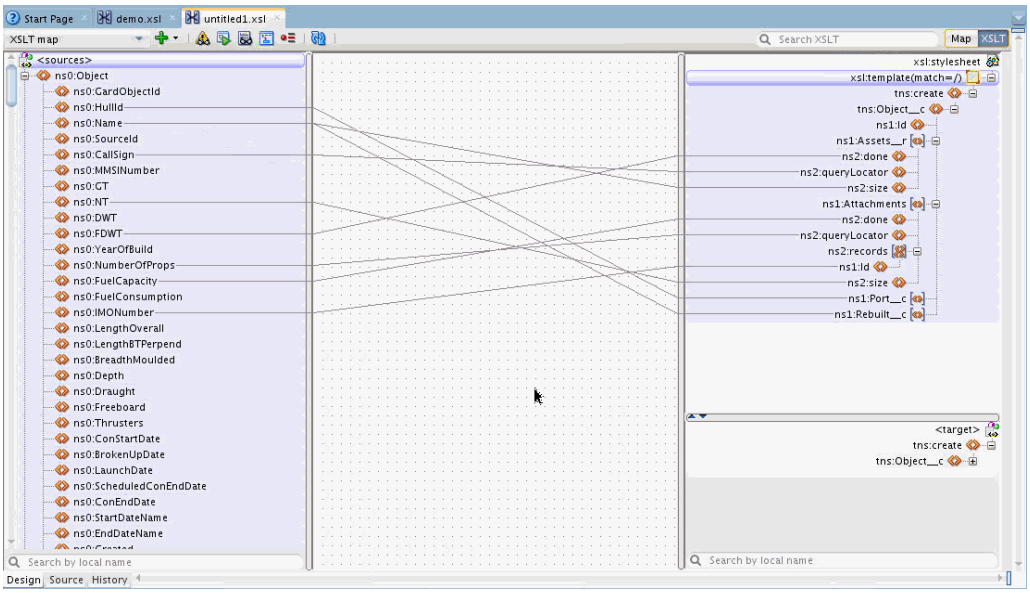
The 12*c* editor contains a new view available within the Design View tab. This is the XSLT View. It can be reached by clicking the XSLT button on the top right of the XSLT editor toolbar.



XSLT View shows the existing statements in the XSLT file. Users who have previously edited XSLT in a source xml editor may appreciate this view. It is organized in the same way as statements would appear in the XSLT source. Using this view will provide a condensed look at the mappings you are creating. For instance, here is a map against a large target schema document in Map View. Note that some lines run off the bottom of the display as they map to nodes that appear in the schema later in the tree.

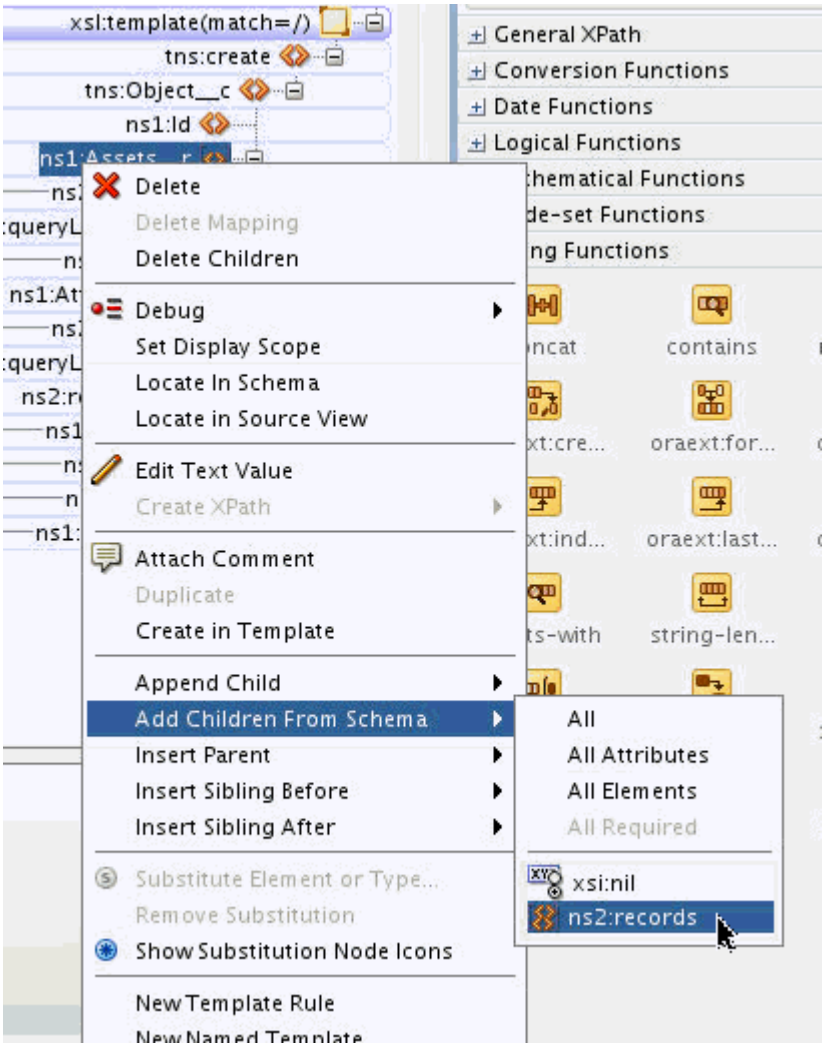


Here is the same mapping in XSLT view:



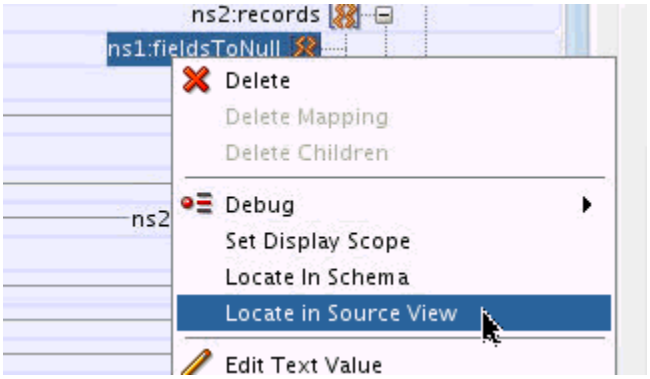
You can now see all of your mappings clearly without unused target nodes taking up space. If you need to add a new target element from the schema, use the **Add Children From Schema** option on the context menu.

From the context menu on any parent node select the Add Children From Schema option and a list of possible child nodes will appear that can be selected and added. You also have the option to select All Attributes/All Elements/All Required from this menu for any parent node.

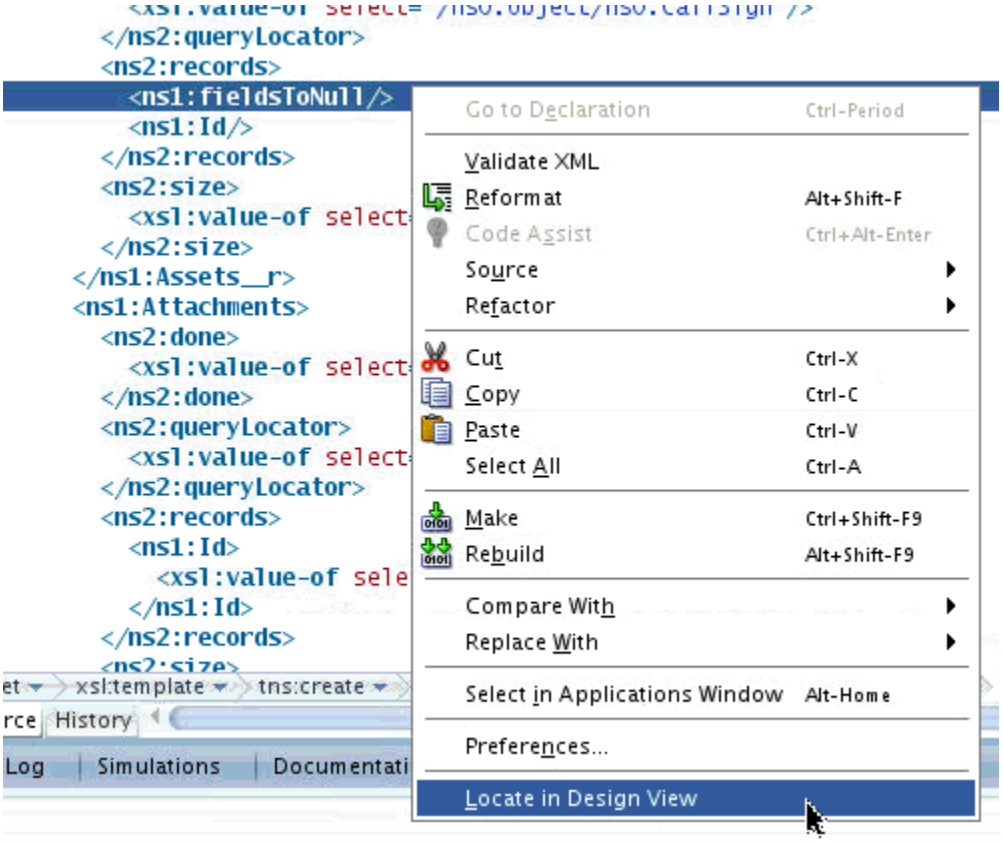


As an added bonus, when nodes are added this way, all required children of any node you insert will be added automatically for you. In the example above, when we select the `ns2:records` element to be added, it is inserted at its correct place in the tree and its required `ns1:id` node is automatically added for you.

If you are used to editing in Source view, an option was added in 12.2.1.0.0 to allow you to move easily back and forth from source to design view. Right-click any node in the XSLT panel and select **Locate in Source View.**



The source view opens and the node is selected:

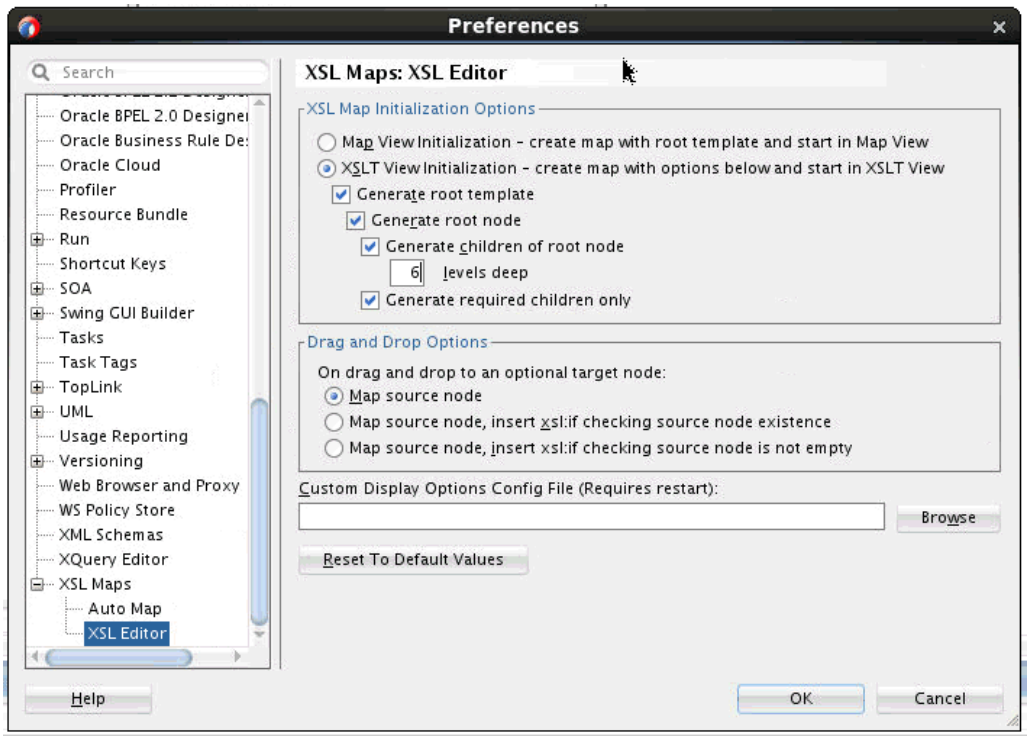


To navigate back to any node in Design view, you can select the **Locate in Design View** option while in Source View.

XSLT view can also be used to insert any XSLT statement and allows the use of named and matched templates (template rules). See Editing an XSLT Map in XSLT View for more information on XSLT view.

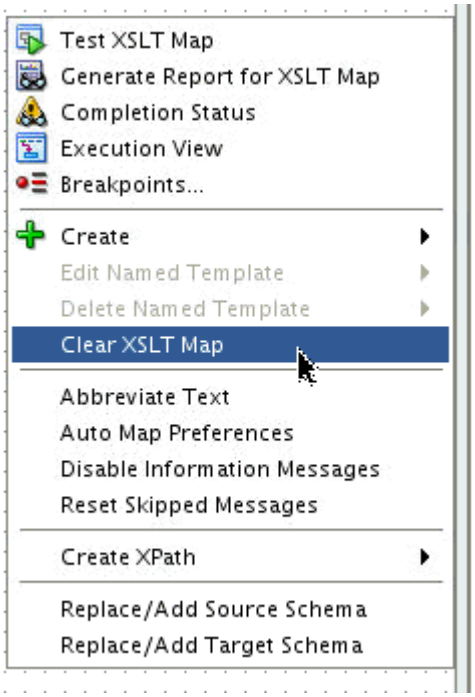
Quick Start for XSLT View

You can set the Preference settings to always start the XSLT Editor in XSLT view. These settings also control the automatic creation of target nodes to get you started. To set the preferences for XSLT View, select **Tools > Preferences** to bring up the Preferences dialog. Then select **XSL Maps > XSL Editor.**



To start in XSLT View, select the **XSLT View Initialization** option with the desired options. If you are working with a large schema, it is a good idea to set a limit to the number of levels of children to be generated.

Then, when you create your XSLT map these options will be used. In addition, these options are used anytime you select the **Clear XSLT Map** option from the canvas context menu.

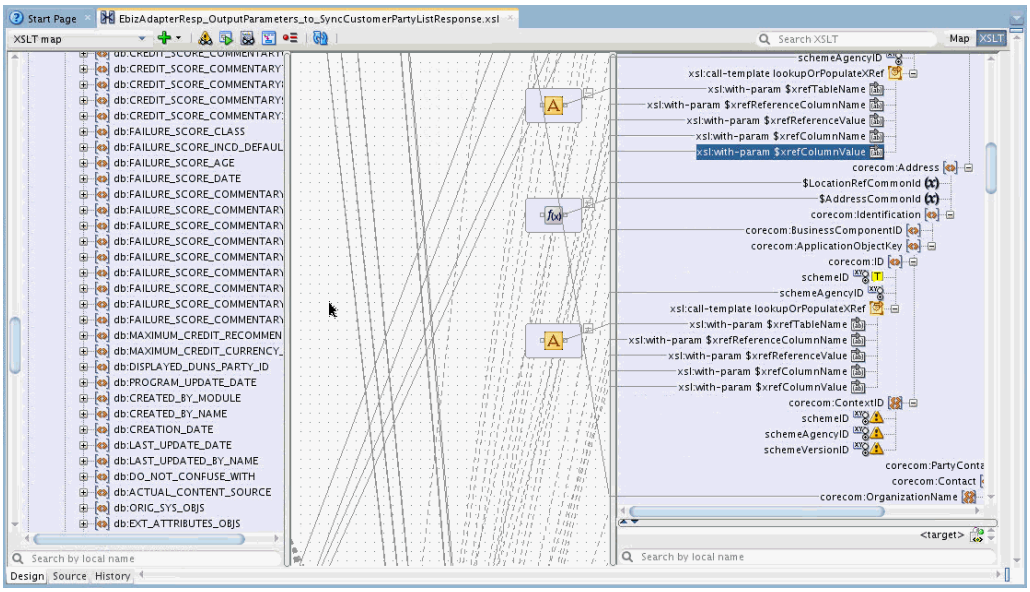


If you do not like your preference settings for a particular map, you can make changes to the preferences and regenerate the initial map by selecting **Clear XSLT Map.**

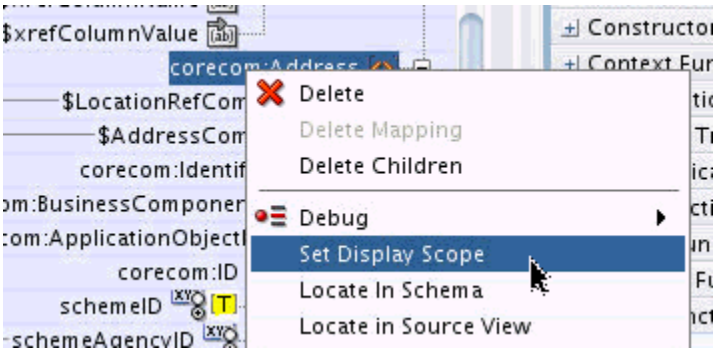
Non-Sparse Mappings

Sometimes, it is necessary to create or modify existing maps that contain large numbers of target elements and consequently large numbers of mappings. When editing a map like this, it can be difficult to keep track of what is going on. For such situations, the 12*c* XSLT Editor has a new feature that enables the user to set the scope of the mapping to show only mappings below a selected target node.

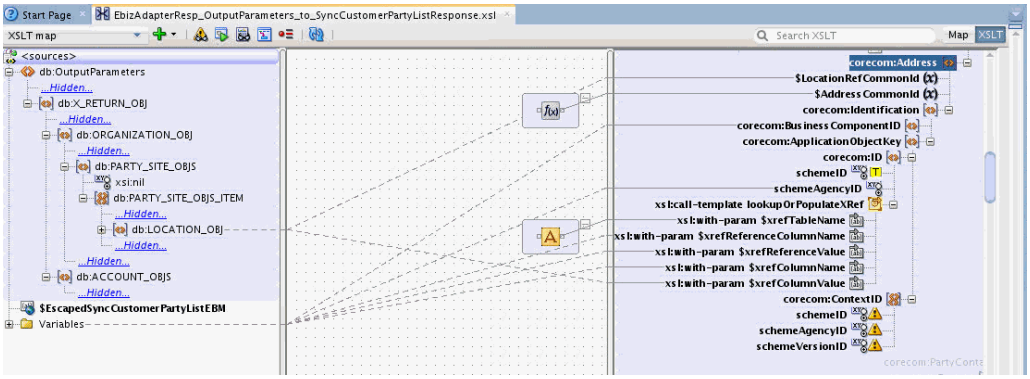
For instance, the following is a non-sparse mapping.



We can set the scope of the mapping to an area in the target tree we would like to work in. Right-click a target node, and select **Set Display Scope.**

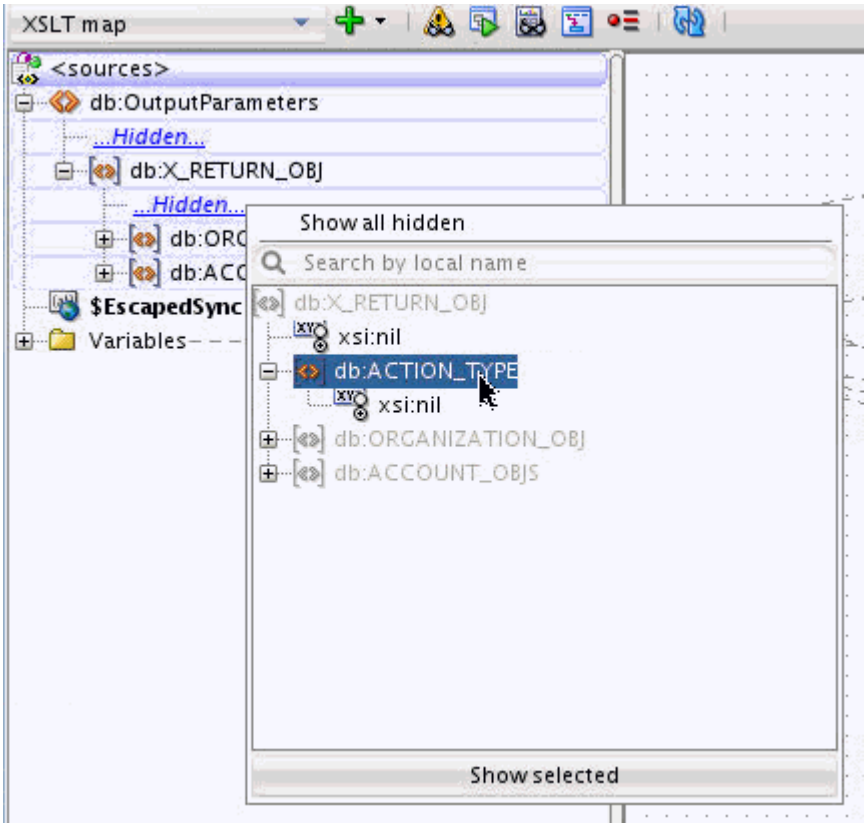


The display is *scoped* to the target node selected. All lines indicating mappings outside of this area are not drawn and the source tree becomes condensed, showing only nodes that are mapped.



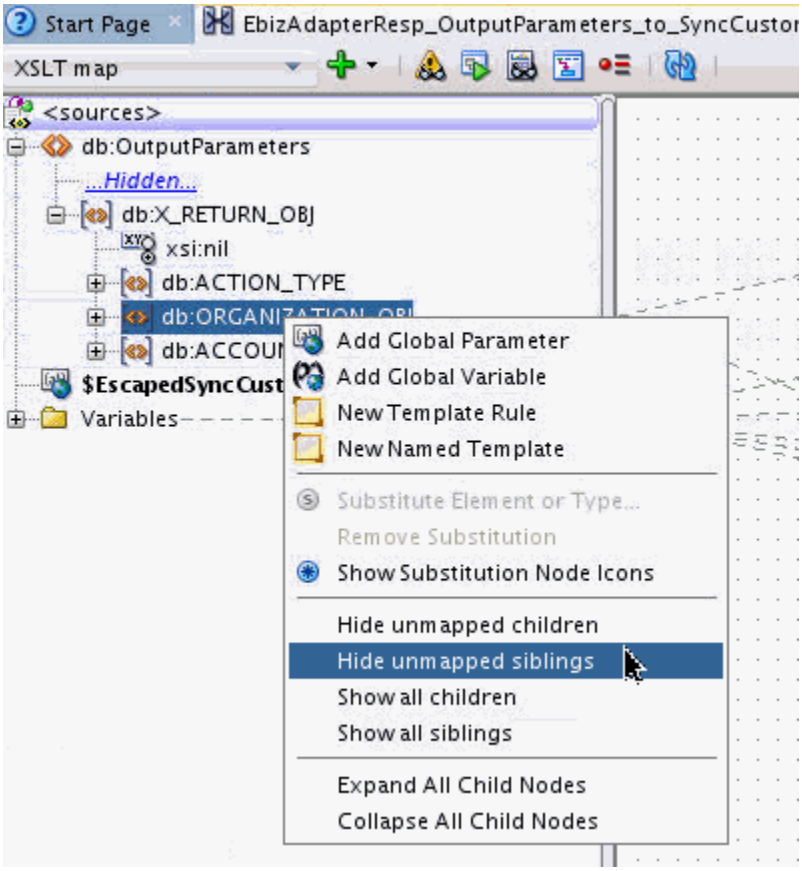
You can then continue to work in the scoped area.

Hidden areas in the source tree can be expanded to show nodes that might be needed for additional mapping. Right-click on any **Hidden** item in the Source tree to see a popup menu with options for searching within the tree and selecting nodes to be shown.

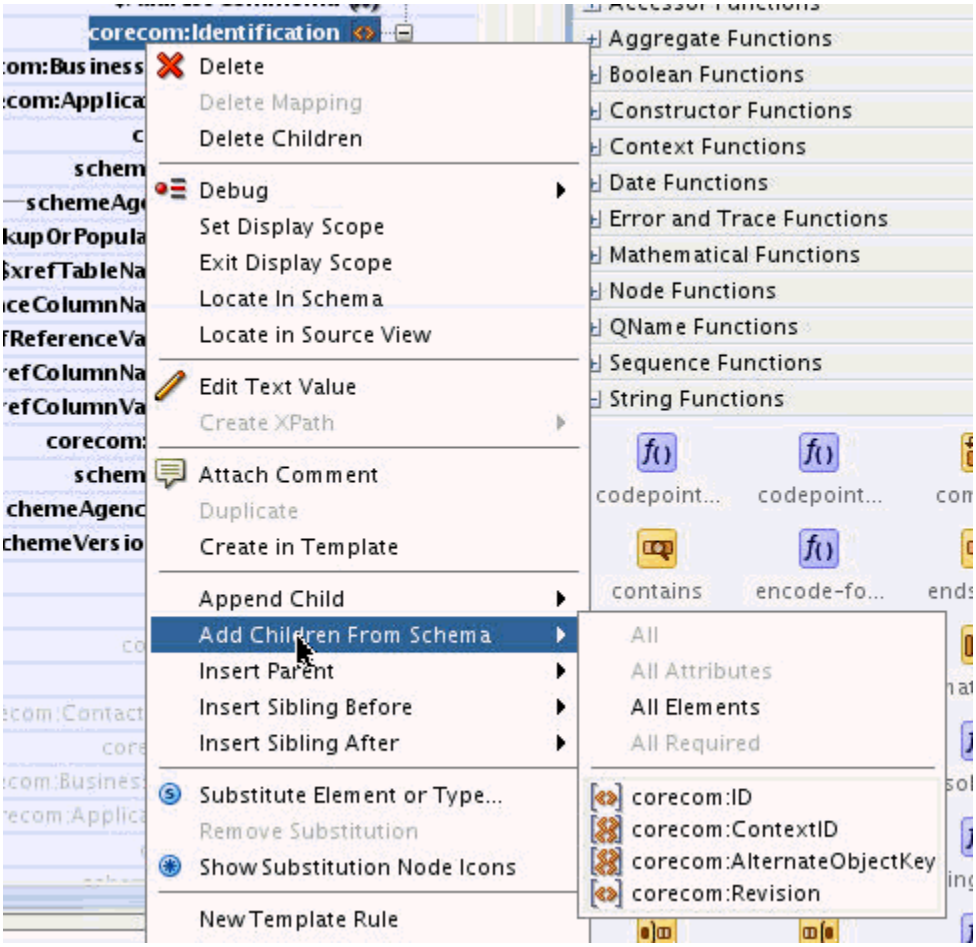


Any search done from this popup will wrap through the tree beyond the currently selected Hidden area, so that you do not have to select the correct Hidden area for the node you are looking for.

There are also options on the main context menu that will hide and expand areas of the source tree. If you right-click on any non-Hidden node in the tree, there are options to show and hide siblings and children of the selected node.



In the target tree, you can add nodes from the schema by using the **Add Children from Schema** option.

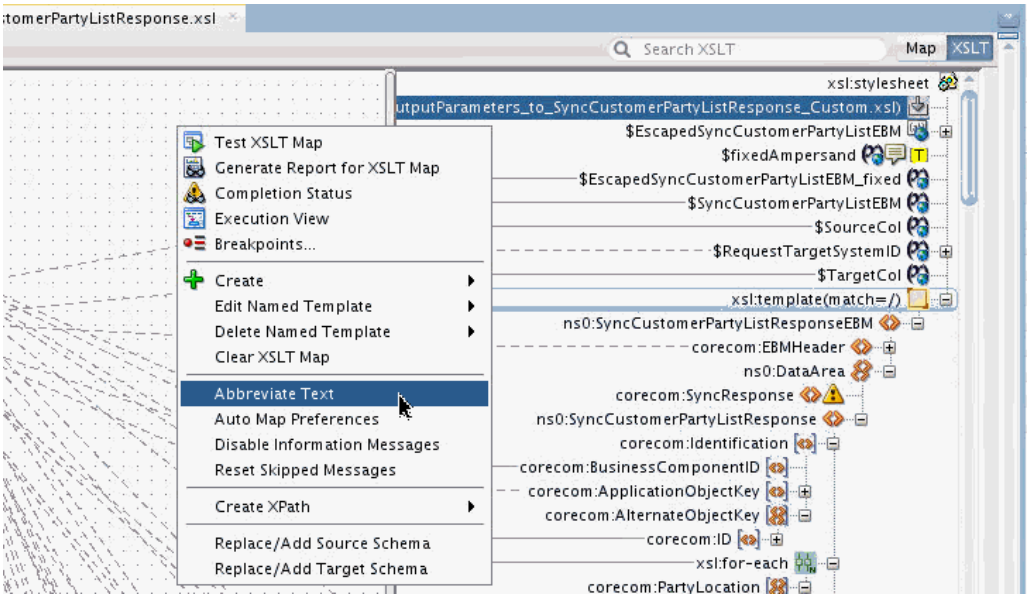


To exit the scoped display, click on a target node outside of the scoped area or select **Exit Display Scope** from the context menu in the target tree.

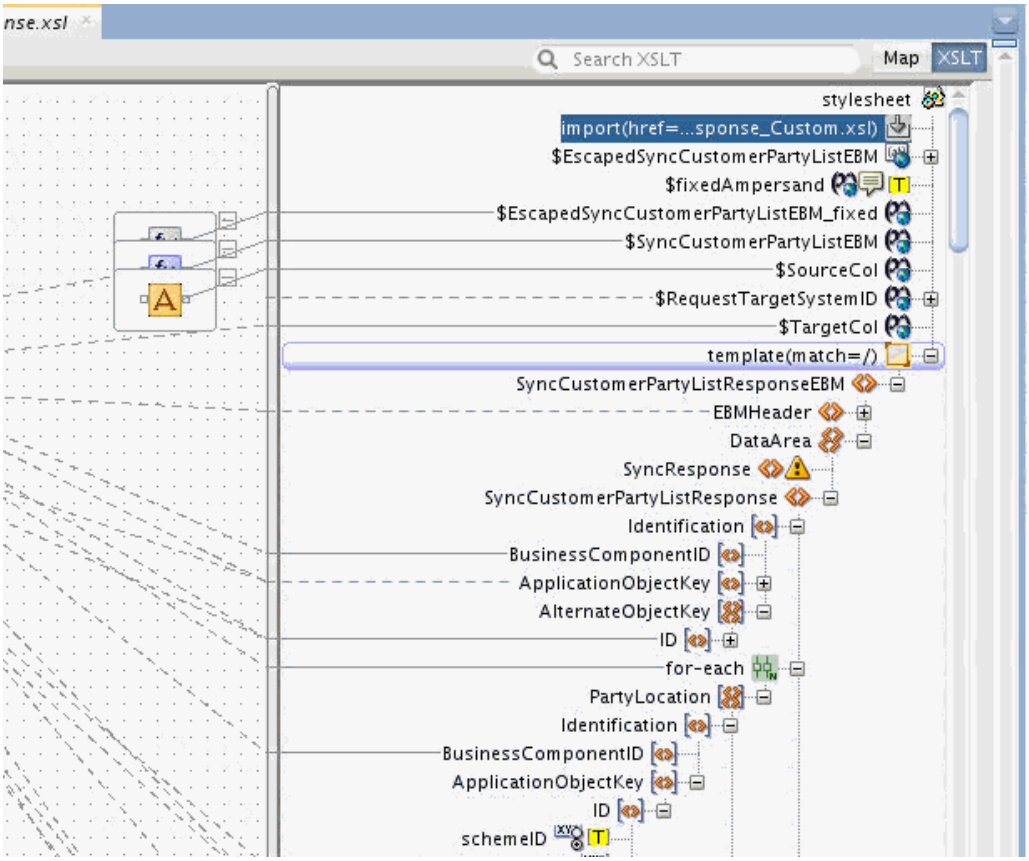
Reducing Textual Clutter

The 12*c* XSLT Editor provides the ability to abbreviate node names and other information in the source and target trees. If you select the **Abbreviate Text** option from the canvas context menu, prefixes will be hidden and the text for certain types of nodes will be abbreviated.

Before abbreviation:



After abbreviation:



You may also create a Custom Display Options Config file where abbreviations for node name text may be defined. For instance, in the example above, the phrase `CustomerPartyList` appears in many node names. This could be abbreviated to `CPL` using a Custom Display Options Config file. Then node names such as `$EscapedSyncCustomerPartyListEBM` will appear as `$EscapedSyncCPLEBM` in the tree.

This does not change the node name in the XSLT or in any XPath statements generated. It only applies to the name that appears on the tree node and can help to reduce overall clutter when schema node definitions use verbose names.

A Custom Display Options Config file can be loaded under XSL Editor preferences. See How to Import a Customization File to Specify Display Preferences in the XSLT Map Editor for more details.

Searching Trees

When searching through large schemas for element names, the search can take long. In 12.1.3, the search does not have a cancel option. This has been added in 12.2.1.

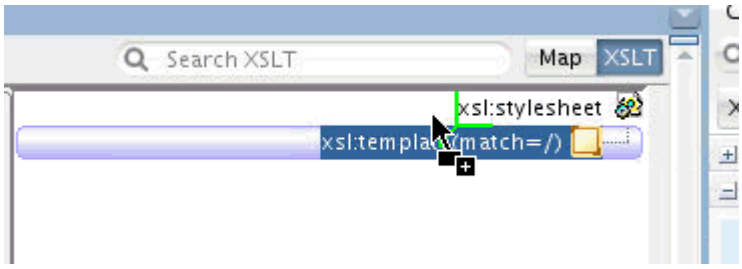
If the search is taking too long, the tree size can be truncated by reducing the **Expansion Depth** for the trees in preferences. Go to **Tools > Preferences.** Select **XSL Maps** from the navigator. Click the **Show Advanced Options** button and change the **Expansion Depth** for the **XML Schema Maximum Expansion Depth** option to a much smaller value. For trees that have hundreds of children at a single level, this value needs to be around 10 levels. This will mean that the search will not go below the level set here, but some trees can contain millions of nodes and the search can take long in that event.

Copying and Modifying a Large Input Document

A user may be tempted to try to copy an input XML document by using the automap feature of the XSLT editor. However, automap generates specific XSLT statements for every node in the schema. On large schemas, this is not an efficient way to copy an input document. This can generate XSLT files that are many MBs in size, and these will be slow to load and difficult to edit. In addition, if the user's mapping is sparse, generating thousands of lines of XSLT to execute against nodes that are defined in the schema, but will never exist at runtime is inefficient.

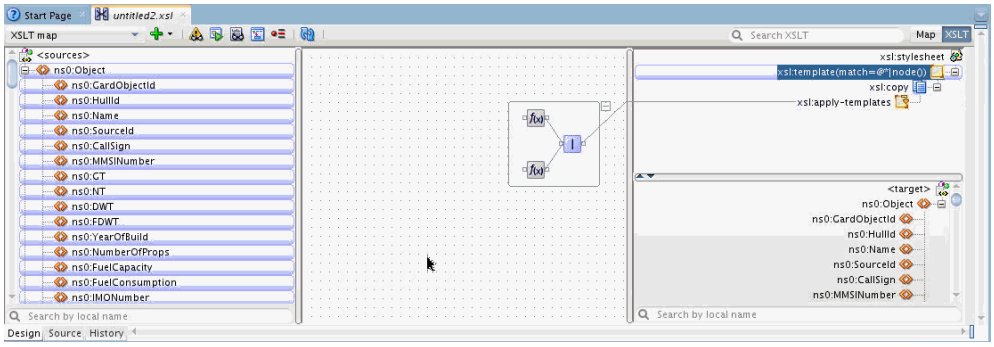
The 12*c* XSLT Editor now supports the creation of matched templates (template rules). In particular, you can now add an identity template that can copy all nodes in the source tree. Use the following steps:

1. Switch to XSLT view.
2. Select XSLT Templates from the Components Window.
3. Drag and drop the Identity Template from the Miscellaneous Templates section to the left side of the `xsl:stylesheet` node. You will see a green highlight when the drop is in the correct position indicating that the template will be added as a child of the stylesheet node.



4. Delete the original root `match="/"` template from the XSLT.

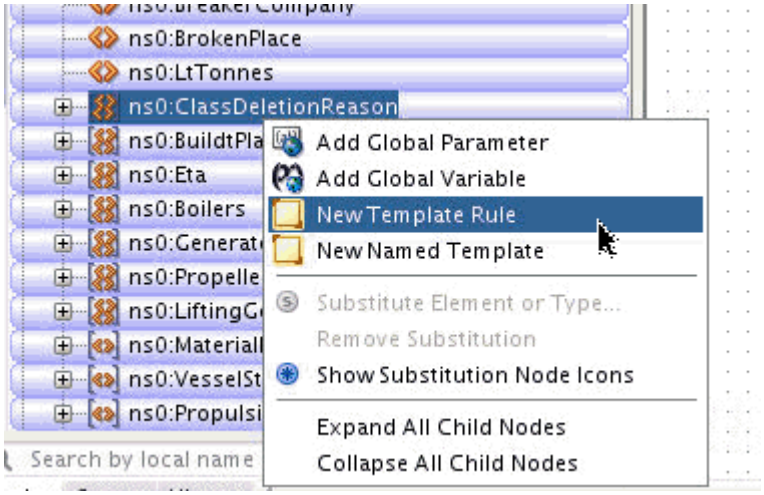
   Your display would look something like the following:



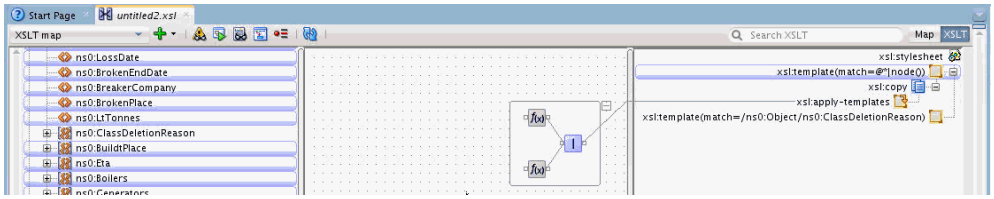
Every node in the input document will be processed by the identity template. This is indicated by the bubble highlighting on each node of the source tree that indicates the context nodes for the selected template. When each node is processed, the `xsl:copy` statement will execute to copy the node to the output. The `apply-templates` statement tells the processor to continue processing any child nodes of the current context node being processed. This will then copy the entire input document.

Additional templates can then be added to make modifications to the tree while it is being copied. For instance, suppose we need to simply remove a node from the input tree. We can do this by adding an empty template for the node we want to remove from the output. In other words, when we process this node, we will output nothing, which will effectively remove the node from the tree.

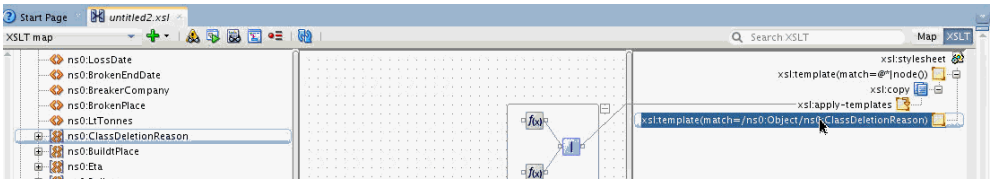
To add an empty template for a node, right-click the source node and select **New Template Rule.**



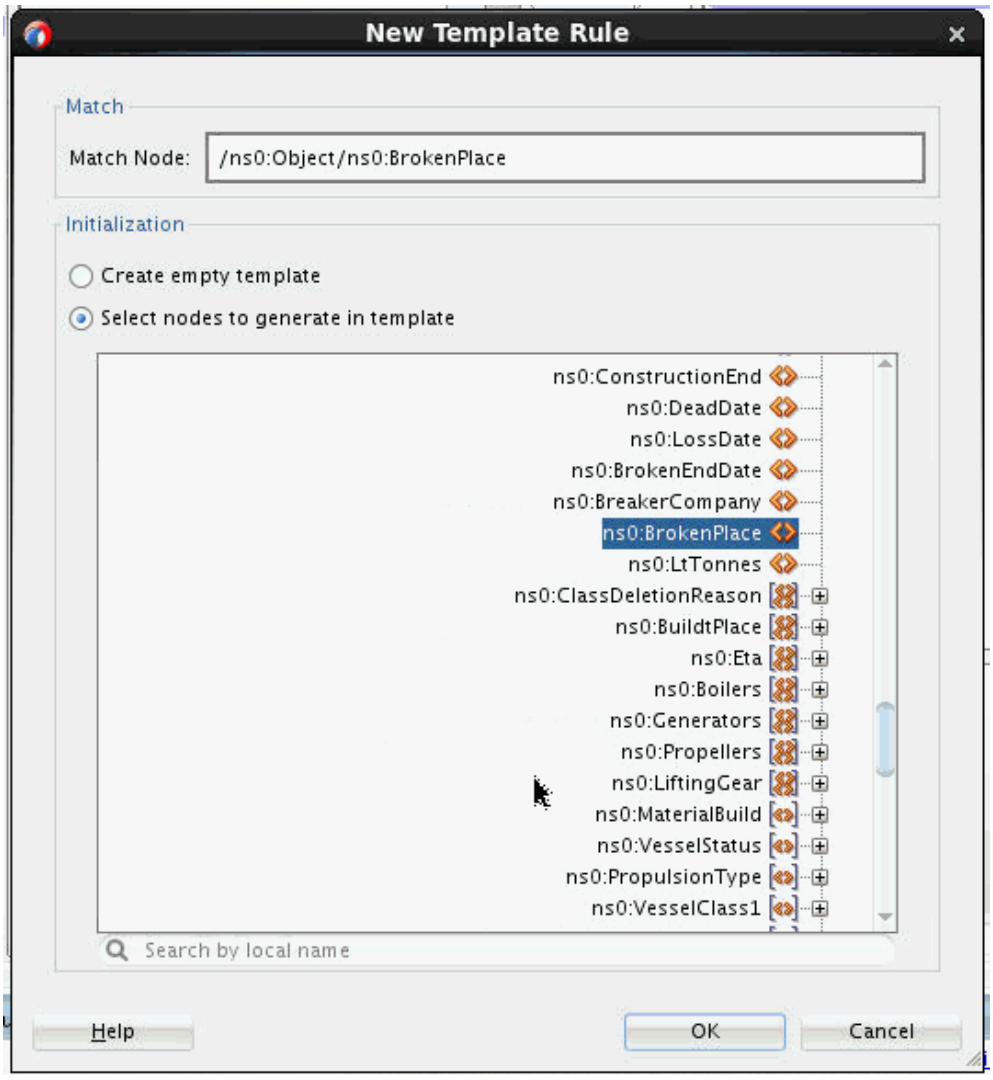
Click **OK** on the New Template Dialog that follows. The template is added. Note the difference in the display for the node when the identity template is selected. It is no longer bubble highlighted, indicating it is no longer processed by the identity template.



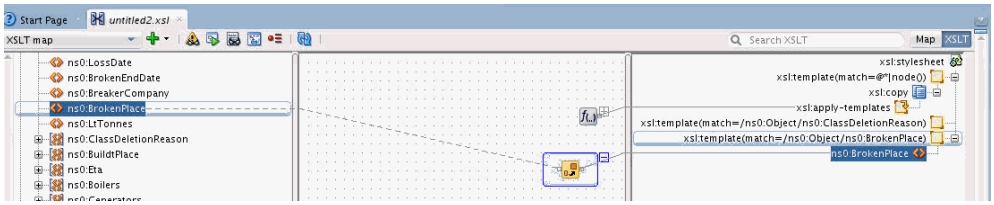
Selecting the new empty template will now highlight this node, showing that this template will process the node and output nothing when it is processed.



Now, suppose we also want to upper-case some text in another node. We can create another template to explicitly process that node to perform the upper-case. We create a template with the **New Template Rule** option selected from the `BrokenPlace` node in the source tree. When the New Template Rule dialog appears, we select the node we want to create in the template.



Click **OK** and the template and the node will be created. We then assign an upper-case function to the node.



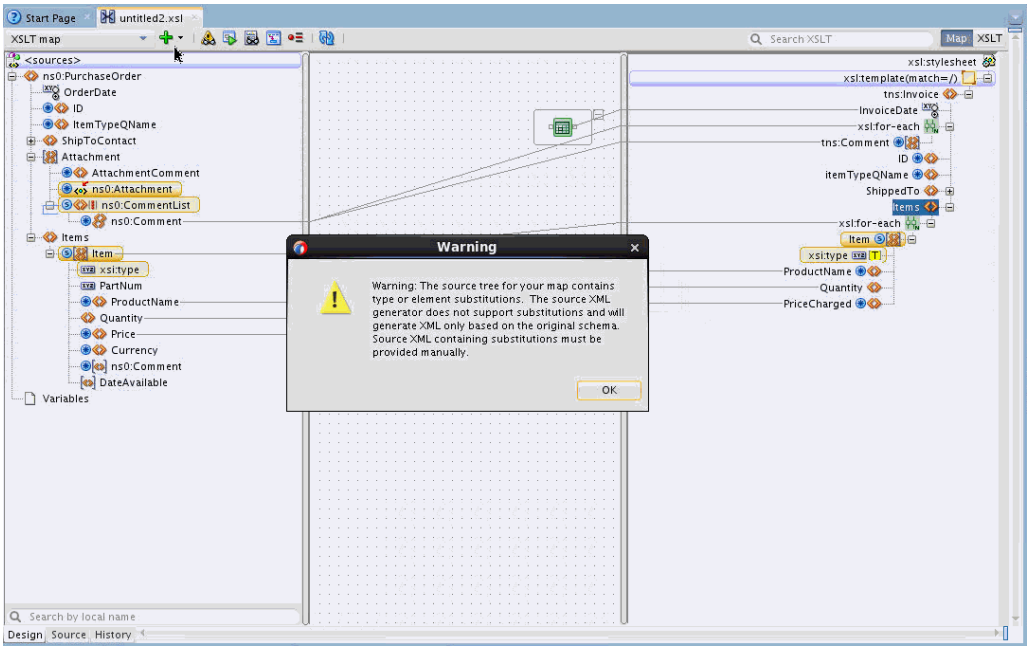
When this node is created in the output, its text will be upper-cased.

In this manner, you can copy and modify a large input XML with very few XSLT statements.

Generating Test Files with Element and Type Substitutions

It is possible in the XSLT editor to perform element and type substitutions based on derived types and substitution groups defined in the XSD. Many schemas contain abstract elements or types that can be overloaded with elements from substitution groups or elements from derived types using `xsi:type`.

The test tool in the XSLT editor does not currently support generation of input XML documents that contain substituted elements. So, when you invoke an XSLT map where substitutions have been made in the source tree, the following warning occurs.



The user then has to modify the input document generated or provide their own test input document. This can become problematic, as the user must make the substitutions themselves in the input document with the correct syntax for the `xsi:type` definition or element substitution needed. This can be more problematic in large schemas where multiple substitutions have been made.

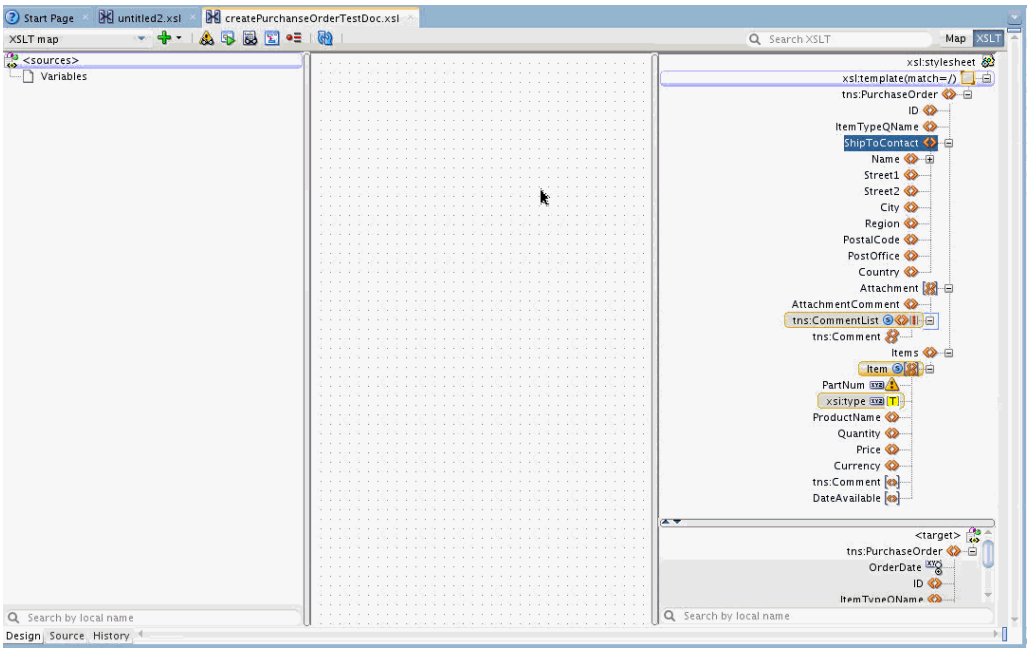
Using the XSLT editor, we can generate an input document that contains the correctly substituted elements with all of the appropriate namespaces/prefixes defined for us. This will provide us with a template for the input test document.

In the mapping above there are two substitutions done in the source tree. The first is a substituted element `CommentList` from a substitution group defined the schema document. The second is an `Item` type substitution for a derived item type defined in the schema document.

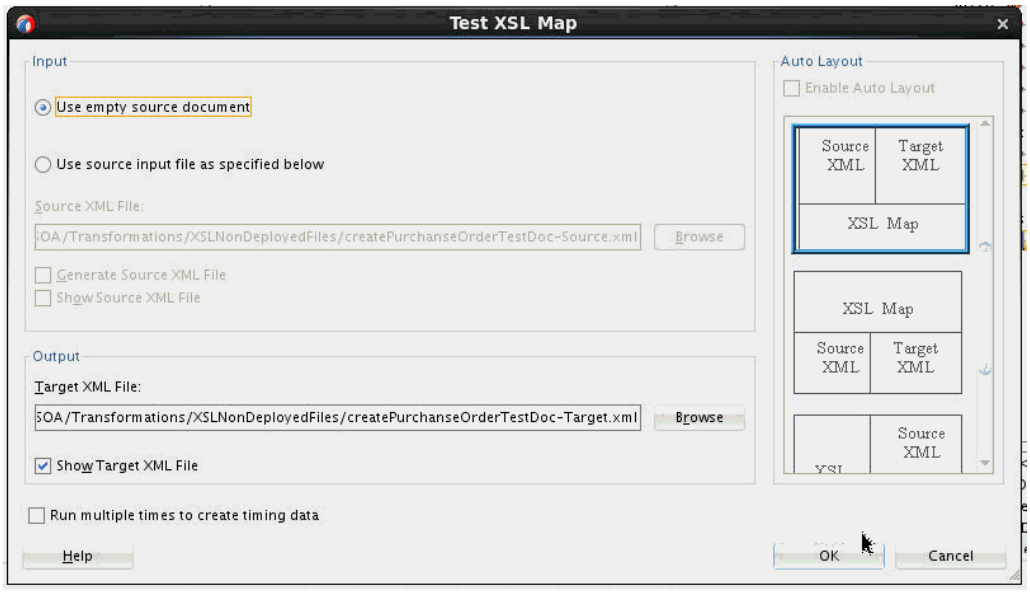
We would like to write a small XSLT map that will generate a document we can use as a test input file for testing this map. It has to contain the correct `xsi:type` and element substitution information defined in our source document.

We create a new map, selecting no schema document for the source and selecting the `PurchaseOrder` schema for the target, as we want to output a `PurchaseOrder` document we can use as test input for the `PurchaseOrder` source in our existing map.

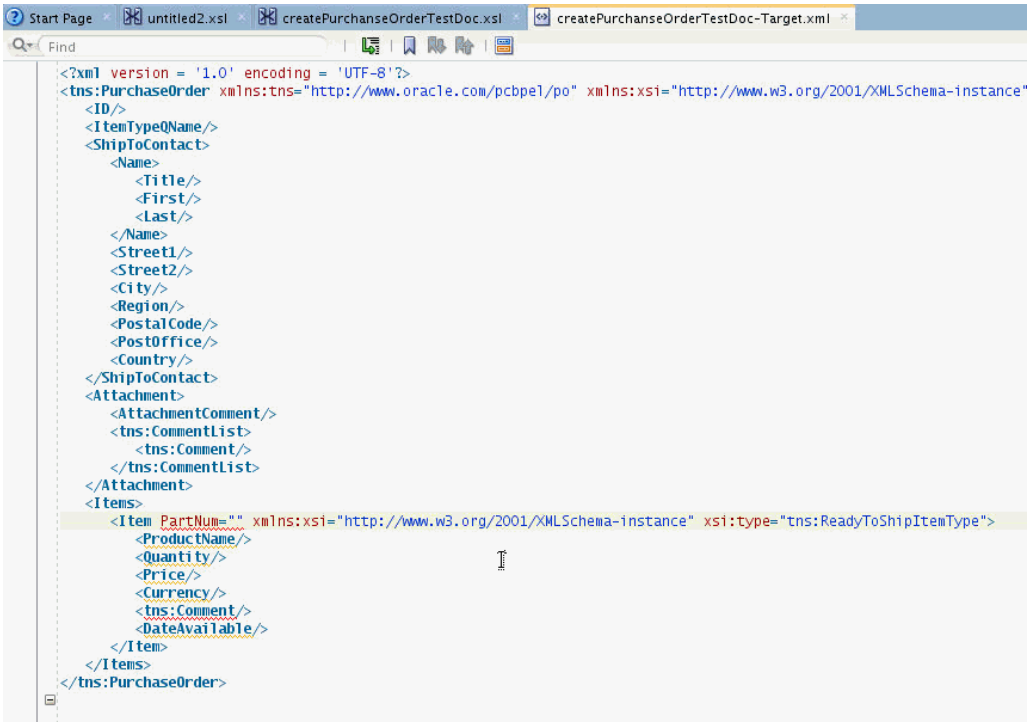
By using **Add Child From Schema** and performing the same substitutions on the `PurchaseOrder` target that we have on our `PurchaseOrder` source, we can create a map that looks like the following:



We then execute this XSLT with the test tool to create our `PurchaseOrder` template document.



This generates a template for our test input document with the correct substitutions for our test.



Appropriate test data can then be entered in the fields defined. Alternatively, you can define data values in the XSLT that generates the test file to pre-populate the test file.