Oracle® E-Business Suite

Integrated SOA Gateway Developer's Guide

Release 12.1

Part No. E12065-13

February 2022

ORACLE

Oracle E-Business Suite Integrated SOA Gateway Developer's Guide, Release 12.1

Part No. E12065-13

Primary Author: Melody Yang

Contributor: Rekha Ayothi, Neeraj Chauhan, Anil Kemisetti, Saritha Nalagandla, Surya Narayana Nellepalli, Nadakuditi Ravindra, Dilbaghsingh Sardar, Vijayakumar Shanmugam, Shivdas Tomar, Abhishek Verma

Contents

4 Using Java Bean Services as REST Services

5 Using XML Gateway Inbound and Outbound Interfaces

6 Using Business Events Through Subscription Model

7 Using Concurrent Programs

Send Us Your Comments

Oracle E-Business Suite Integrated SOA Gateway Developer's Guide, Release 12.1

Part No. E12065-13

Oracle welcomes customers' comments and suggestions on the quality and usefulness of this document. Your feedback is important, and helps us to best meet your needs as a user of our products. For example:

- Are the implementation steps correct and complete?
- Did you understand the context of the procedures?
- Did you find any errors in the information?
- Does the structure of the information help you with your tasks?
- Do you need different information or graphics? If so, where, and in what format?
- Are the examples correct? Do you need more examples?

If you find any errors or have any other suggestions for improvement, then please tell us your name, the name of the company who has licensed our products, the title and part number of the documentation and the chapter, section, and page number (if available).

Note: Before sending us your comments, you might like to check that you have the latest version of the document and if any concerns are already addressed. To do this, access the new Oracle E-Business Suite Release Online Documentation CD available on My Oracle Support and www.oracle.com. It contains the most current Documentation Library plus all documents revised or released recently.

Send your comments to us using the electronic mail address: appsdoc_us@oracle.com

Please give your name, address, electronic mail address, and telephone number (optional).

If you need assistance with Oracle software, then please contact your support representative or Oracle Support Services.

If you require training or instruction in using Oracle software, then please contact your Oracle local office and inquire about our Oracle University offerings. A list of Oracle offices is available on our Web site at www.oracle.com.

Preface

Intended Audience

Welcome to Release 12.1 of the *Oracle E-Business Suite Integrated SOA Gateway Developer's Guide.*

This guide assumes you have a working knowledge of the following:

- The principles and customary practices of your business area.
- Computer desktop application usage and terminology.
- Oracle E-Business Suite integration interfaces.
- B2B, A2A and BP integrations.

This documentation assumes familiarity with Oracle E-Business Suite. It is written for the technical consultants, implementers and system integration consultants who oversee the functional requirements of these applications and deploy the functionality to their users.

If you have never used Oracle E-Business Suite, we suggest you attend one or more of the Oracle E-Business Suite training classes available through Oracle University.

See Related Information Sources on page xii for more Oracle E-Business Suite product information.

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.

Structure

Related Information Sources

This book is included in the Oracle E-Business Suite Documentation Library. If this guide refers you to other Oracle E-Business Suite documentation, use only the latest Release 12.1 versions of those guides.

Online Documentation

All Oracle E-Business Suite documentation is available online (HTML or PDF).

- **Online Help** - Online help patches (HTML) are available on My Oracle Support.
- **Oracle E-Business Suite Documentation Library** - This library, which is included in the Oracle E-Business Suite software distribution, provides PDF documentation as of the time of each release.
- **Oracle E-Business Suite Documentation Web Library** - This library, available on the Oracle Help Center (https://docs.oracle.com/cd/E18727_01/index.htm), provides the latest updates to Oracle E-Business Suite Release 12.1 documentation. Most documents are available in PDF and HTML formats.

- **Release Notes** - For information about changes in this release, including new features, known issues, and other details, see the release notes for the relevant product, available on My Oracle Support.

- **Oracle Electronic Technical Reference Manual -** The Oracle Electronic Technical Reference Manual (eTRM) contains database diagrams and a detailed description of database tables, forms, reports, and programs for each Oracle E-Business Suite product. This information helps you convert data from your existing applications and integrate Oracle E-Business Suite data with non-Oracle applications, and write custom reports for Oracle E-Business Suite products. The Oracle eTRM is available from My Oracle Support.

Related Guides

You should have the following related books on hand. Depending on the requirements of your particular installation, you may also need additional manuals or guides.

Oracle Alert User's Guide

This guide explains how to define periodic and event alerts to monitor the status of your Oracle E-Business Suite data.

Oracle Cloud Using the Oracle E-Business Suite Adapter with Oracle Integration Cloud

This guide describes how to set up and use Oracle E-Business Suite Adapter connections in Oracle Integration to access supported Oracle E-Business Suite interfaces and REST services as inbound or outbound integrations from Oracle E-Business Suite.

As part of the integration documents in Oracle Cloud Platform as a Service (PaaS), this guide is available in Oracle Cloud Documentation.

Oracle E-Business Suite Concepts

This book is intended for all those planning to deploy Oracle E-Business Suite Release 12, or contemplating significant changes to a configuration. After describing the Oracle E-Business Suite architecture and technology stack, it focuses on strategic topics, giving a broad outline of the actions needed to achieve a particular goal, plus the installation and configuration choices that may be available.

Oracle E-Business Suite CRM System Administrator's Guide

This manual describes how to implement the CRM Technology Foundation (JTT) and use its System Administrator Console.

Oracle E-Business Suite Desktop Integration Framework Developer's Guide

Oracle E-Business Suite Desktop Integration Framework is a development tool that lets you define custom integrators for use with Oracle Web Applications Desktop Integrator. This guide describes how to define and manage integrators and all associated supporting objects, as well as how to download and upload integrator definitions.

Oracle E-Business Suite Developer's Guide

This guide contains the coding standards followed by the Oracle E-Business Suite development staff. It describes the Oracle Application Object Library components needed to implement the Oracle E-Business Suite user interface described in the *Oracle E-Business Suite User Interface Standards for Forms-Based Products*. It provides information to help you build your custom Oracle Forms Developer forms so that they integrate with Oracle E-Business Suite. In addition, this guide has information for customizations in features such as concurrent programs, flexfields, messages, and logging.

Oracle E-Business Suite Flexfields Guide

This guide provides flexfields planning, setup, and reference information for the Oracle E-Business Suite implementation team, as well as for users responsible for the ongoing maintenance of Oracle E-Business Suite product data. This guide also provides information on creating custom reports on flexfields data.

Oracle Application Framework Developer's Guide

This guide contains the coding standards followed by the Oracle E-Business Suite development staff to produce applications built with Oracle Application Framework. This guide is available in PDF format on My Oracle Support and as online documentation in Oracle JDeveloper 10g with Oracle Application Extension.

Oracle Application Framework Personalization Guide

This guide covers the design-time and runtime aspects of personalizing applications built with Oracle Application Framework.

Oracle E-Business Suite Mobile Apps Administrator's Guide, Release 12.1 and 12.2

This guide describes how to set up an Oracle E-Business Suite instance to support connections from Oracle E-Business Suite mobile apps. It also describes common administrative tasks for configuring Oracle E-Business Suite mobile apps and setup tasks for enabling push notifications for supported mobile apps. Logging and troubleshooting information is also included in this book.

Oracle E-Business Suite Mobile Apps Developer's Guide, Release 12.1 and 12.2

This guide describes how to develop enterprise-distributed mobile apps by using mobile application archive (MAA) files and how to implement corporate branding. It also explains required tasks on implementing push notifications for supported mobile apps. In addition, it includes how to implement Oracle E-Business Suite REST services to develop custom mobile apps by using the Login component from Oracle E-Business Suite Mobile Foundation or using any mobile app development framework if desired.

Oracle E-Business Suite Installation Guide: Using Rapid Install

This book is intended for use by anyone who is responsible for installing or upgrading Oracle E-Business Suite. It provides instructions for running Rapid Install either to carry out a fresh installation of Oracle E-Business Suite Release 12, or as part of an upgrade from Release 11i to Release 12. The book also describes the steps needed to install the technology stack components only, for the special situations where this is applicable.

Oracle Fusion Middleware Adapter for Oracle Applications User's Guide

This book covers the use of Oracle E-Business Suite Adapter (formerly known as "Adapter for Oracle Applications" in Oracle Applications Server 10g or Oracle Fusion Middleware 11g releases) in developing integrations between Oracle E-Business Suite and trading partners.

This book is available in the Oracle Application Server 10g Documentation Library and Oracle Fusion Middleware 11g Documentation Library.

Oracle E-Business Suite System Administrator's Guide Documentation Set

This documentation set provides planning and reference information for the Oracle E-Business Suite System Administrator. *Oracle E-Business Suite System Administrator's Guide - Configuration* contains information on system configuration steps, including defining concurrent programs and managers, enabling Oracle Applications Manager features, and setting up printers and online help. *Oracle E-Business Suite System Administrator's Guide - Maintenance* provides information for frequent tasks such as monitoring your system with Oracle Applications Manager, administering Oracle E-Business Suite Secure Enterprise Search, managing concurrent managers and reports, using diagnostic utilities including logging, managing profile options, and using alerts. *Oracle E-Business Suite System Administrator's Guide - Security* describes User Management, data security, function security, auditing, and security configurations.

Oracle E-Business Suite User's Guide

This guide explains how to navigate, enter data, query, and run reports using the user interface (UI) of Oracle E-Business Suite. This guide also includes information on setting user profiles, as well as running and reviewing concurrent requests.

Oracle E-Business Suite User Interface Standards for Forms-Based Products

This guide contains the user interface (UI) standards followed by the Oracle E-Business Suite development staff. It describes the UI for the Oracle E-Business Suite products and how to apply this UI to the design of an application built by using Oracle Forms.

Oracle Diagnostics Framework User's Guide

This manual contains information on implementing and administering diagnostics tests for Oracle E-Business Suite using the Oracle Diagnostics Framework.

Oracle E-Business Suite Integrated SOA Gateway User's Guide

This guide describes the high level service enablement process, explaining how users can browse and view the integration interface definitions and services residing in Oracle Integration Repository.

Oracle E-Business Suite Integrated SOA Gateway Implementation Guide

This guide explains how integration repository administrators can manage and administer the web service activities for integration interfaces including native packaged integration interfaces, composite services (BPEL type), and custom integration interfaces. It also describes how to invoke web services from Oracle E-Business Suite by employing the Oracle Workflow Business Event System, and how to manage web service security, configure logs, and monitor SOAP messages.

Oracle e-Commerce Gateway User's Guide

This guide describes the functionality of Oracle e-Commerce Gateway and the necessary setup steps in order for Oracle E-Business Suite to conduct business with trading partners through Electronic Data Interchange (EDI). It also contains how to run extract programs for outbound transactions, import programs for inbound transactions, and the relevant reports.

Oracle e-Commerce Gateway Implementation Manual

This guide describes implementation details, highlighting additional setup steps needed for trading partners, code conversion, and Oracle E-Business Suite. It also provides architecture guidelines for transaction interface files, troubleshooting information, and a description of how to customize EDI transactions.

Oracle iSetup Developer's Guide

This manual describes how to build, test, and deploy Oracle iSetup Framework interfaces.

Oracle Workflow Administrator's Guide

This guide explains how to complete the setup steps necessary for any product that includes workflow-enabled processes. It also describes how to manage workflow processes and business events using Oracle Applications Manager, how to monitor the progress of runtime workflow processes, and how to administer notifications sent to workflow users.

Oracle Workflow Developer's Guide

This guide explains how to define new workflow business processes and customize existing Oracle E-Business Suite-embedded workflow processes. It also describes how to define and customize business events and event subscriptions.

Oracle Workflow User's Guide

This guide describes how users can view and respond to workflow notifications and monitor the progress of their workflow processes.

Oracle Workflow API Reference

This guide describes the APIs provided for developers and administrators to access Oracle Workflow.

Oracle Workflow Client Installation Guide

This guide describes how to install the Oracle Workflow Builder and Oracle XML Gateway Message Designer client components for Oracle E-Business Suite.

Oracle XML Gateway User's Guide

This guide describes Oracle XML Gateway functionality and each component of the Oracle XML Gateway architecture, including Message Designer, Oracle XML Gateway Setup, Execution Engine, Message Queues, and Oracle Transport Agent. It also explains how to use Collaboration History that records all business transactions and messages exchanged with trading partners.

The integrations with Oracle Workflow Business Event System, and the Business-to-Business transactions are also addressed in this guide.

Integration Repository

The Oracle Integration Repository is a compilation of information about the service endpoints exposed by the Oracle E-Business Suite of applications. It provides a complete catalog of Oracle E-Business Suite's business service interfaces. The tool lets users easily discover and deploy the appropriate business service interface for integration with any system, application, or business partner.

The Oracle Integration Repository is shipped as part of the Oracle E-Business Suite. As your instance is patched, the repository is automatically updated with content appropriate for the precise revisions of interfaces in your environment.

Do Not Use Database Tools to Modify Oracle E-Business Suite Data

Oracle STRONGLY RECOMMENDS that you never use SQL*Plus, Oracle Data Browser, database triggers, or any other tool to modify Oracle E-Business Suite data unless otherwise instructed.

Oracle provides powerful tools you can use to create, store, change, retrieve, and maintain information in an Oracle database. But if you use Oracle tools such as SQL*Plus to modify Oracle E-Business Suite data, you risk destroying the integrity of your data and you lose the ability to audit changes to your data.

Because Oracle E-Business Suite tables are interrelated, any change you make using an Oracle E-Business Suite form can update many tables at once. But when you modify Oracle E-Business Suite data using anything other than Oracle E-Business Suite, you may change a row in one table without making corresponding changes in related tables. If your tables get out of synchronization with each other, you risk retrieving erroneous information and you risk unpredictable results throughout Oracle E-Business Suite.

When you use Oracle E-Business Suite to modify your data, Oracle E-Business Suite automatically checks that your changes are valid. Oracle E-Business Suite also keeps track of who changes information. If you enter information into database tables using database tools, you may store invalid information. You also lose the ability to track who has changed your information because SQL*Plus and other database tools do not keep a record of changes.

1

Oracle E-Business Suite Integrated SOA Gateway Overview

Oracle E-Business Suite Integrated SOA Gateway Overview

Building on top of Oracle Fusion Middleware and service-oriented architecture (SOA) technology, Oracle E-Business Suite Integrated SOA Gateway (ISG) is a complete set of service infrastructure to provide, consume, and administer Oracle E-Business Suite web services.

With service enablement feature, integration interfaces published in the Oracle Integration Repository can be transformed into SOAP and REST based web services.

SOAP-based services are described in WSDLs and are deployed to the application server for service consumption. REST services described in WADLs are used for user-driven applications such as Oracle E-Business Suite mobile applications.

Oracle E-Business Suite Integrated SOA Gateway provides Service Invocation Framework to invoke and consume web services provided by other applications.

For more information about each integration interface and service, see the *Oracle E-Business Suite Integrated SOA Gateway User's Guide*; for more information on implementing and administering Oracle E-Business Suite Integrated SOA Gateway, see the *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

Major Features

Oracle E-Business Suite Integrated SOA Gateway contains the following features:

- Display all Oracle E-Business Suite integration interface definitions through Oracle Integration Repository
- Support custom integration interfaces from Oracle Integration Repository
- Provide service enablement capability (SOAP and REST services) for seeded and

custom integration interfaces within Oracle E-Business Suite

- Use the Integration Repository user interface to perform design-time activities such as generate and deploy Oracle E-Business Suite web services
- Support synchronous interaction pattern for both SOAP-based and REST-based web services

  Note: In this release, only PL/SQL APIs, Concurrent Programs, and Business Service Objects can be exposed as both SOAP and REST services. Java Bean Services, Application Module Services, Open Interface Tables, and Open Interface Views can be exposed as REST services only.

- Support multiple authentication types for inbound service requests in securing web service content
- Enforce function security and role-based access control security to allow only authorized users to execute administrative functions
- Provide centralized, user-friendly user interface for logging configuration
- Audit and monitor Oracle E-Business Suite service operations from native SOA Monitor
- Leverage Oracle Workflow Business Event System to enable web service invocation from Oracle E-Business Suite

Major Components Features and Definitions

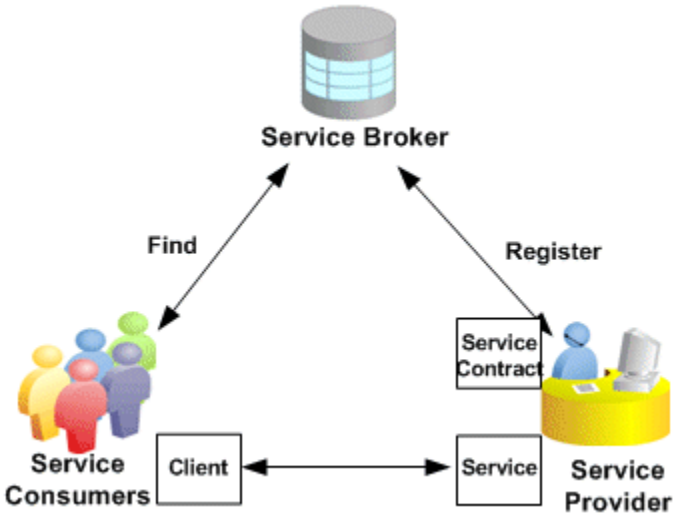
The better understand Oracle E-Business Suite Integrated SOA Gateway and its key components, this section describes some key features and the definition of each component.

Native Service Enablement

Service enablement is the key feature within Oracle E-Business Suite Integrated SOA Gateway. It provides a mechanism that allows native packaged integration interface definitions residing in Oracle Integration Repository to be further transformed into web services that comply with web standards. Additionally, these services can be deployed from the Integration Repository to the application server allowing more consumptions over the web.

To understand the basic concept of web services and how the service works, the following diagram illustrates the essential components for service enablement:

Major Components for Service Enablement



A Service Provider is the primary engine underlying the web services. It facilitates the service enablement for various types of interfaces.

A Service Consumer (web service client) is the party that uses or consumes the services provided by the Service Provider.

A Service Broker (Service Registry) describes the service's location and contract to ensure service information is available to any potential service consumer.

Composite Services

Composite services use the native service as building blocks to construct the sequence of business flows. Basically, this interface type orchestrates the invocation sequence of discrete web services into a meaningful end-to-end business process through a web service composition language BPEL (business process execution language).

For example, use Oracle BPEL Process Manager (BPEL PM) to integrate the Order-to-Receipt business process that contains sales order entry, item availability check, pack and ship, and invoice to Accounts Receivable sub processes handled by various applications. This approach effectively tightens up the control of each individual process and makes the entire business flow more efficiently.

Oracle Integration Repository and Service Enablement

Oracle Integration Repository, an integral part of Oracle E-Business Suite, is the centralized repository that contains numerous interface endpoints exposed by applications within the Oracle E-Business Suite.

To effectively manage all integration interfaces and services incurred within the Oracle E-Business Suite, Oracle E-Business Suite Integrated SOA Gateway now supports complex business processes or composite services, web service generation and

deployment, as well as business event subscriptions through the centralized Integration Repository.

You can browse these interface definitions and services through the Oracle Integration Repository user interfaces. Users with administrator privileges can further perform administrative tasks through the same interfaces.

Oracle Integration Repository supports the following interface types:

- PL/SQL
- XML Gateway
- Concurrent Programs
- Business Events
- Open Interface Tables/Views
- EDI
- Business Service Object (Service Beans)
- Java

  Apart from normal Java APIs, Java interface includes the following subcategories:

  - Application Module Services

    Note: Application Module Implementation class is a Java class that provides access to business logic governing the OA Framework based components and pages. Such Java classes are called Application Module Services and are categorized as a subtype of Java interface.

  - Java Bean Services

    Note: Java APIs whose methods use parameters of either simple data types or serializable Java Beans are categorized as Java Bean Services. Such Java APIs can be exposed as REST-based web services.

  - Security Services

    Note: Security Services are a set of predefined and pre-deployed REST services from Oracle Application Object

Library. These services include Authentication and Authorization services for mobile applications. These services are built on Java; therefore, they are categorized as a subtype of Java interface.

Note that Java APIs for Forms web services are desupported in Oracle E-Business Suite Release 12.2. If you are planning to use this type of interfaces as web services, you are advised to use alternate serviceable interfaces, such as PL/SQL and Business Service Objects interfaces, which can be deployed as web services. Refer to My Oracle Support Knowledge Document 966982.1 for the suggested alternatives to the existing Java APIs for Forms services.

- Composite Services

Service Invocation Framework

To invoke all integration services from Oracle E-Business Suite, Oracle E-Business Suite Integrated SOA Gateway uses the Service Invocation Framework (SIF) that leverages Oracle Workflow Java Business Event System (JBES) and a seeded Java rule function to allow any WSDL-described service to be invoked.

By using this service invocation framework, developers or implementors can interact with web services through WSDL descriptions instead of working directly with SOAP APIs, the usual programming model. This approach lets you use WSDL as a normalized description of disparate software, and allows you to access this software in a manner that is independent of protocol or location.

Since this feature is the major development framework in invoking web services within the entire Oracle E-Business Suite, detailed implementation information is described in a separate chapter in this book.

See Web Service Invocation Using Service Invocation Framework, page 12-2.

SOA Monitor

SOA Monitor is a centralized, light-weight service execution monitoring and management tool. It not only monitors all the SOAP requests that SOA Provider and Web Service Provider process, but also provides auditing feature for the SOAP messages if the auditing feature is enabled.

With SOA Monitor, the Integration Repository Administrator can effectively manage and identify errors incurred during the service deployment life cycle and take necessary actions to expedite the interaction between services.

Manage Security

Oracle E-Business Suite integrated SOA Gateway enforces the security rules through subject authentication and authorization:

- To authenticate users who request Oracle E-Business Suite web services, request messages must be checked based on the selected authentication type:
  - The SOAP messages must be authenticated using UsernameToken or SAML Token based security. The identified authentication information is embedded in the `wsse:security` Web Security headers.
  - The REST messages are authenticated using HTTP Basic Authentication or Token Based Authentication at HTTP or HTTPS transport level.
- To authorize users on specific services or operations, the access permissions must be explicitly given to the users through security grants. Multiple organization access control (MOAC) security rule is also implemented for authorizing interface execution related to multiple organizations.

Additionally, input message header (such as SOAHeader for SOAP services or RESTHeader for REST services) is used to pass application contexts needed in invoking Oracle E-Business Suite services as part of the subject authorization.

2

Discovering and Viewing Integration Interfaces

Overview

Similar to regular users or system integration analysts, system integration developers can view integration interfaces and their details from Oracle Integration Repository, as well as review generated or deployed Web service WSDL files in the appropriate Web Service region. The developers cannot perform administrative tasks, such as generating or deploying Web services, which are done by the integration repository administrators.

However, the developers have more privileges than the analysts in viewing all types of integration interfaces including public, private, and internal interface types from Oracle Integration Repository. These privileges allow developers to have sufficient integration interface information which could be useful to better understand each integration interface from different perspectives.

> **Note:** System integration analysts can view *Public* integration interfaces only, and they do not have the access privileges to view *Private to Application* and *Internal to Oracle* interfaces from the Oracle Integration Repository.

This section covers the following topics:

- Understanding REST Messages, page 2-41

Searching and Viewing Integration Interfaces

To better understand each integration interface and the integration between different applications, Oracle E-Business Suite Integrated SOA Gateway allows system integration developers and integration repository administrators to have more interface access privileges in viewing all integration interface types regardless of public, private, or internal interface types.

Browsing the Integration Interfaces

When viewing integration interfaces, you can browse by product family, by interface type, or by standard based on your selection in the View By drop-down list. Expand the navigation tree in one of these views to see a list of the available interfaces.

For more information on how to browse the interfaces, see Browsing the Integration Interfaces, *Oracle E-Business Suite Integrated SOA Gateway User's Guide*.

Searching the Integration Interfaces

To search for an integration interface, click **Search** to access the main Search page. After clicking the **Show More Search Options** link in the Search page, you can find *Private to Application* and *Internal to Oracle* interface types along with *Public* and *All* displayed from the Scope drop-down menu. If 'All' is selected from the Scope field, then all integration interfaces including public, private to application, and internal to Oracle interfaces can be listed in the results region.

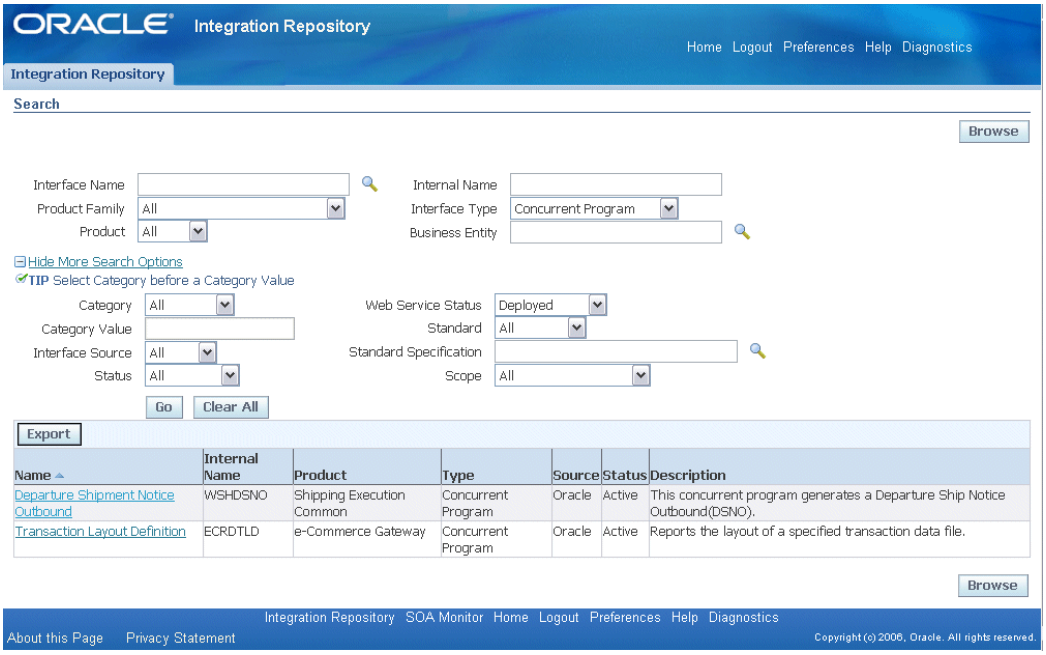
> **Note:** System integration analysts can view *Public* integration interfaces only, and they do not have the access privileges to view *Private to Application* and *Internal to Oracle* interfaces from the Oracle Integration Repository.
>
> In addition, they can only find 'All' (default) and 'Public' list of values available from the Scope drop-down list. And only Public integration interfaces will be retrieved and listed in the search result even if they do not change the default value 'All' in the Scope field.
>
> For detailed information on Public, Private to Application, and Internal to Oracle, see Scope, *Oracle E-Business Suite Integrated SOA Gateway User's Guide*.

By using the search feature, you can easily locate a deployed web service for a particular product or product family if you want to use the deployed service for a partner link creation while orchestrating the BPEL process.

For example, to locate all deployed web services for concurrent program, first select 'Concurrent Program' from the Interface drop-down list and then click **Show More Search Options** to select 'Deployed' for the Web Service Status field. After executing the

search, you should find all deployed web services for the concurrent program interface type.

Search Page for Searching Deployed Web Services



Searching for Java Bean Services, Application Module Services, and Security Services

Java Bean Services, Application Module Services, and Security Services are all specialized Java classes and are categorized as a *subtype* of Java interfaces and displayed in the Integration Repository under the Java interface type.

To easily locate these interfaces or services through the Search page, click the **Show More Search Options** link to display more search fields. Enter the following key search values along with any product family or scope if needed as the search criteria:

- Category: Interface Subtype
- Category Value: 'Java APIs for Forms', 'Java Bean Services', 'Application Module Services', or 'Security Services'

> **Note:** Although you can search and locate the 'Java APIs for Forms' interfaces from the search, these Forms-based web services are desupported in Oracle E-Business Suite Release 12.2. If you are planning to use this type of interfaces as web services, you are advised to use alternate serviceable interfaces, such as PL/SQL and Business Service Objects interfaces, which can be deployed as web services. Refer to My Oracle Support Knowledge Document 966982.1 for the suggested alternatives to the existing Java APIs for Forms services.

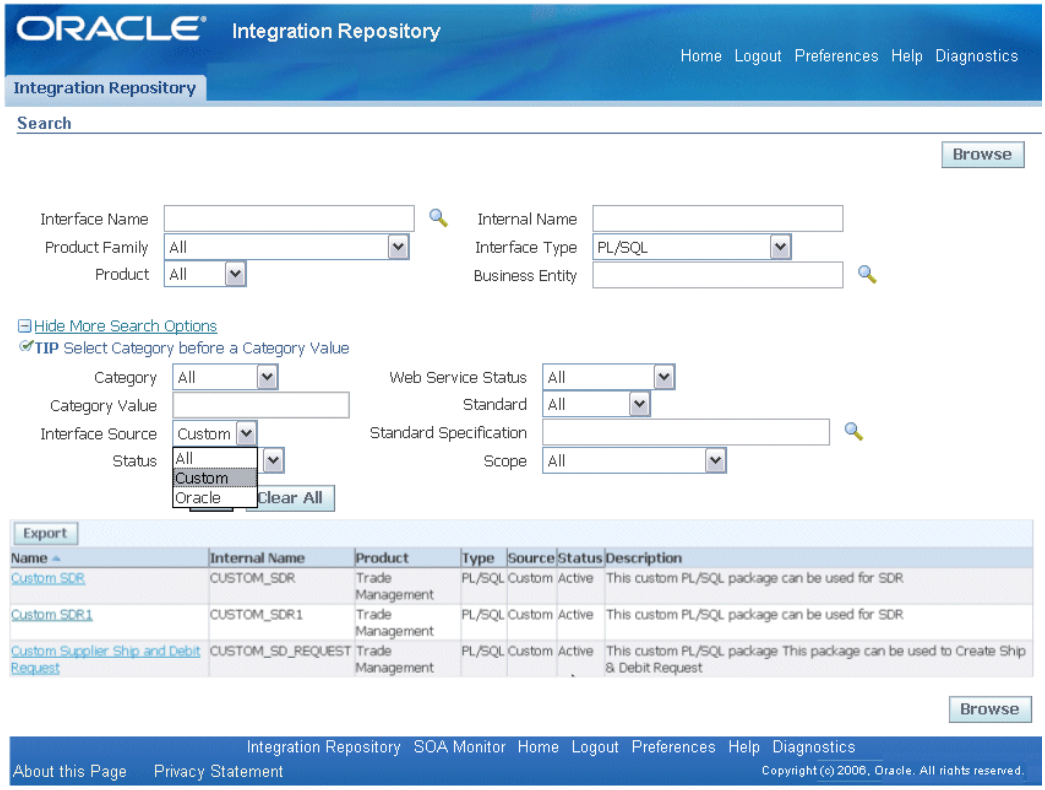
To view the interface or service details, click the interface or service name link that you want to view from the search result region. The interface details page is displayed. For more information on interface details, see Reviewing Interface Details, page 2-5.

Searching for Custom Integration Interfaces

Annotated custom interface definitions, once they are uploaded successfully, are merged into the interface types they belong to and displayed together with Oracle interfaces from the Integration Repository browser window. To easily distinguish annotated custom interface definitions from Oracle interfaces, the Interface Source "Custom" is used to categorize those custom integration interfaces in contrast to Interface Source "Oracle" for Oracle interfaces.

Therefore, you can search for custom integration interfaces by clicking **Show More Search Options** to display more search fields.

Search Page for Searching Custom Integration Interfaces



Enter the following information along with any interface type, product family, or scope if needed as the search criteria:

- Interface Source: Custom

For information on how to view custom integration interfaces, see Viewing Custom Integration Interfaces, *Oracle E-Business Suite Integrated SOA Gateway User's Guide*.

For more information on each search field in the Search page, see Searching for an Integration Interface, *Oracle E-Business Suite Integrated SOA Gateway User's Guide*.

To search for all integration interface types:

1. Log in to Oracle Integration Repository as a user who has the System Integration Developer role.

   Select the Integrated SOA Gateway responsibility from the navigation menu. Select the Integration Repository link to open the repository browser.

2. Click **Search** to open the main Search page.

3. Enter appropriate search information such as product family, product, interface type, or business entity.

4. Click **Show More Search Options** to open more search options.

   - To search custom integration interfaces, select 'Custom' in the Interface Source field.

   - To search Java Bean Services, Application Module Services, or Security Services, select 'Interface Subtype' in the Category field and select 'Java Bean Services', 'Application Module Services' or 'Security Services' in the Category Value field.

5. To view deployed integration interfaces, select 'Deployed' from the Web Service Status field drop-down list.

6. To view all integration interfaces, select All from the Scope field. This allows all integration interfaces including Public, Internal to Oracle, and Private to Application displayed in the results region.

7. To view integration interfaces of Public, Internal to Oracle, or Private to Application type, select 'Public', 'Internal to Oracle', or 'Private to Application' from the Scope drop-down list respectively.

8. Click **Go** to execute the search. All interfaces that match your search criteria are displayed.

9. Select an interface type from the search result to view the interface details.

Reviewing Interface Details

After searching for an integration interface, a system integration developer can review a selected interface details by clicking on an interface name from the search result page. This opens the interface details page where the developer can view the interface general information, a description region, a source region, and an interface methods or procedure and functions region.

Based on the selected interface, the developer can view the associated web service information including the SOAP-based and REST-based services if it's available in the interface details page.

> **Note:** In this release, only PL/SQL APIs, Concurrent Programs, and Business Service Objects can be exposed as both SOAP and REST services. Java Bean Services, Application Module Services, Open Interface Tables, and Open Interface Views can be exposed as REST services only.

For more information on SOAP-based services, see Common Information on SOAP Web Services, *Oracle E-Business Suite Integrated SOA Gateway User's Guide*.

For more information on REST-based services, see Common Information on REST Web Services, *Oracle E-Business Suite Integrated SOA Gateway User's Guide*.

Once a web service is generated, a system integration developer can then use the associated WSDL or WADL definition in invoking the Oracle E-Business Suite service. For information on how to invoke Oracle E-Business Suite services, see each individual chapter described in this book.

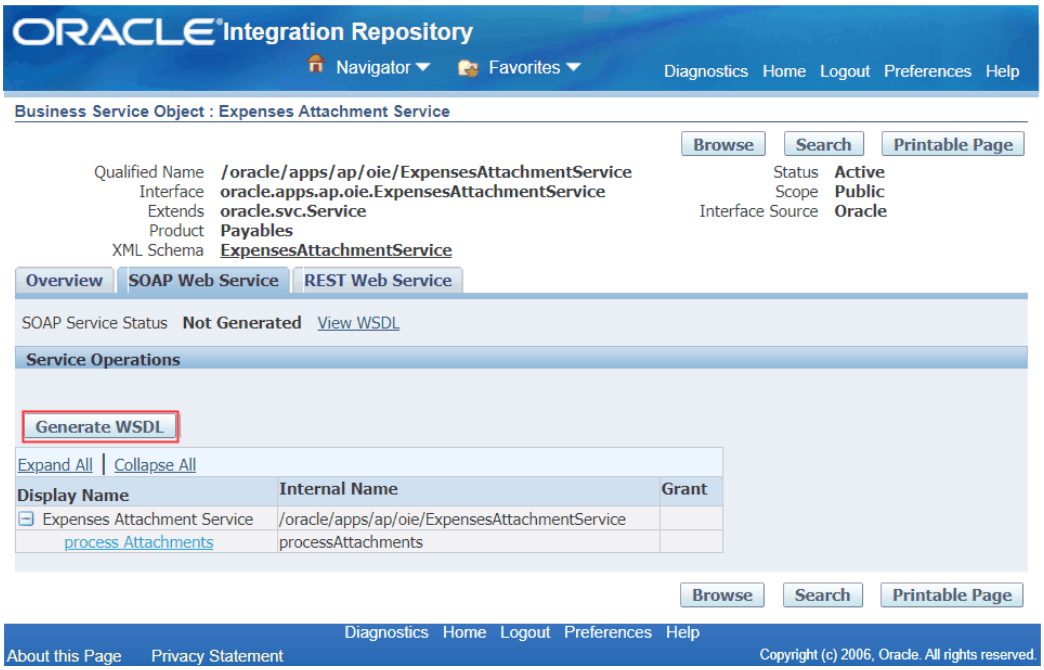
Generating SOAP Web Services

Users who have the System Integration Developer role can transform interface definitions into SOAP web services represented in WSDL description.

Generating SOAP Services

In the SOAP Web Service tab (or the Web Service region for an XML Gateway interface), a system integration developer can click **Generate** (or **Generate WSDL** for an XML Gateway or a Business Service Object interface) to generate a SOAP service represented in WSDL.

Business Service Object Interface Details Page with "Generate WSDL" Button Highlighted in the SOAP Web Service Tab



After Service Generation

After a SOAP service has been successfully generated, the SOAP Service Status is changed from 'Not Generated' to 'Generated' indicating that the selected interface has WSDL description available, but it has not yet been deployed.

> **Important:** If service generation is still in progress, then 'Generating' is displayed as the SOAP service status.

Click the **View WSDL** link to view the generated WSDL code. For more information about WSDL, see: Reviewing WSDL Element Details, page 2-8.

Regenerating Web Services

If the interface definition is changed, the web service can be regenerated by clicking **Regenerate** (or **Regenerate WSDL** for an XML Gateway or a Business Service Object interface). Upon regeneration, the service definition will also be changed to reflect the changes done in the interface. You need to modify its web service clients based on the new service definition.

If interface definition is not changed, then regenerating the service would not change the service definition. You can continue to use the existing web service clients with the new service definition.

For more information on generating SOAP services, see Generating SOAP Web Services, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

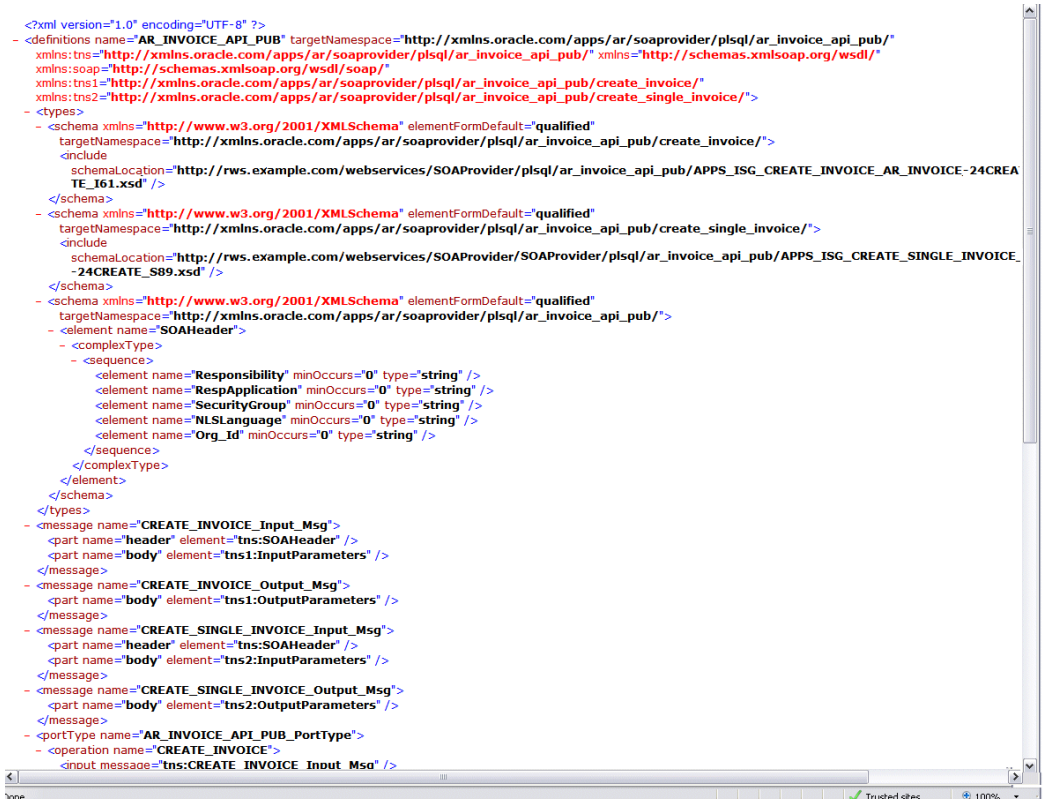
Reviewing WSDL Element Details

If an interface can be exposed as a SOAP service, the corresponding WSDL file is created and can be accessed through the interface details page.

After clicking the **View WSDL** link, you can view the corresponding WSDL description of the generated or deployed SOAP service. This XML-based document describes a selected web service as a set of endpoints operating on messages containing document-oriented information.

For example, click the deployed **View WSDL** link for the "PL/SQL: Invoice Creation" interface to display the corresponding WSDL document:

Deployed SOAP Service WSDL Description



Note: The http:// address in the new window has the exact WSDL URL information that appeared in the interface details page. This address can be copied and used directly in any of the web service clients for invoking the services.

For example, it can be used while creating a partner link for the invocation of the interface that is exposed as a SOAP web service in a

BPEL process.

WSDL Document Structure

A WSDL document is simply a set of definitions. There is a **definitions** element at the root, and definitions inside. The *definitions* element defines the set of services that the web service offers.

It often contains an optional `TargetNamespace` property, a convention of XML schema that enables the WSDL document to refer to itself.

The structure of this definitions element can be like:

```
<definitions name="nmtoken"
  <targetNamespace="uri">
   <import namespace="uri" location="uri"/> *
</definitions>
```

For example, a corresponding WSDL document of the Invoice Creation API (AR_INVOICE_API_PUB) that is exposed as a SOAP service appears in a new window.

```
<definitions name="AR_INVOICE_API_PUB"
targetNamespace="http://xmlns.oracle.
com/apps/ar/soaprovider/plsql/ar_invoice_api_pub/"
xmlns:tns="http://xmlns.oracle.
com/apps/ar/soaprovider/plsql/ar_invoice_api_pub/
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns1="http://xmlns.oracle.
com/apps/ar/soaprovider/plsql/ar_invoice_api_pub/create_invoice/"
xmlns:tns2="http://xmlns.oracle.
com/apps/ar/soaprovider/plsql/ar_invoice_api_pub/create_single_invoice/>
```

For example, the *definitions* element specifies that this WSDL document is the called 'AR_INVOICE_API_PUB'. It also specifies numerous namespaces that will be used throughout the remainder of the document. It also specifies a default namespace: `xmlns=http://schemas.xmlsoap.org/wsdl/`.

In addition to the *definitions* element, web services are defined using the following six major elements:

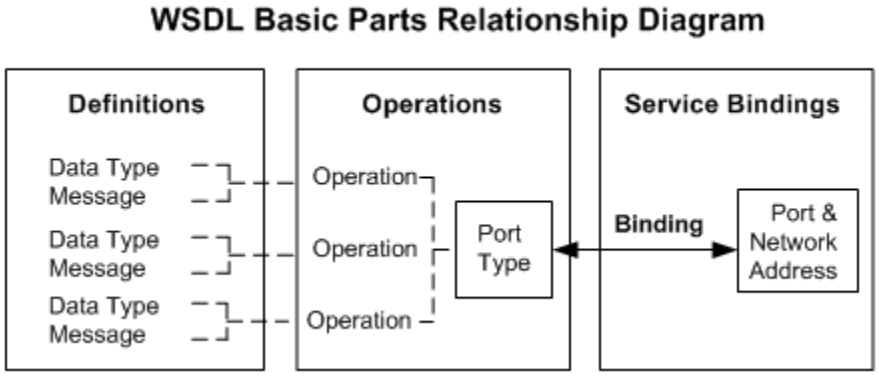
- **Types**: It provides data type definitions used to describe the messages exchanged.
- **Message**: It represents an abstract definition of the data being transmitted.

  A message consists of logical parts, each of which is associated with a definition within some type system.
- **PortType**: It is a set of abstract operations. Each operation refers to an input message and output messages.
- **Binding**: It specifies concrete protocol and data format specifications for the operations and messages defined by a particular portType.
- **Port**: It specifies an address for a binding, thus defining a single communication

endpoint.

- **Service**: It is used to aggregate a set of related ports.

The following diagram shows the relationship of the basic parts of WSDL:



Types

The **types** element contains all data types used in all method calls described in the WSDL. It can be used to specify the XML Schema (xsd:schema) that is used to describe the structure of a WSDL Part.

The structure of this Types element can be like:

```
<definitions...>
 <types>
    <xsd:schema.../>*
   </types>
</definitions>
```

For example, the "Invoice Creation" SOAP service contains the following two functions:

- CREATE_INVOICE
- CREATE_SINGLE_INVOICE

Each function is described in the data type definition. WSDL prefers the use of XSD as the type of system mechanism to define the types in a message schema. As a result, the message schema location of the CREATE_INVOICE function is defined in `APPS_XX_BPEL_CREATE_INVOICE_AR_INVOICE_API_PUB-24CREATE_INV.xsd`. The message schema location of the CREATE_SINGLE_INVOICE function is defined in `APPS_XX_BPEL_CREATE_SINGLE_INVOICE_AR_INVOICE_API_PUB-24CREATE_SIN.xsd`.

```
<types>
   <schema xmlns="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified"
   targetNamespace="http://xmlns.oracle.
com/apps/ar/soaprovider/plsql/ar_invoice_api_pub/create_invoice/">
   <include
   schemaLocation="https://<hostname>:
<port>/webservices/SOAProvider/plsql/ar_invoice_api_pub/APPS_XX_BPEL_CRE
ATE_INVOICE_AR_INVOICE_API_PUB-24CREATE_INV.xsd"/>
   </schema>
  <schema xmlns="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified"
   targetNamespace="http://xmlns.oracle.
com/apps/ar/soaprovider/plsql/ar_invoice_api_pub/create_single_invoice/"
>
   <include
   schemaLocation="https://<hostname>:
<port>/webservices/SOAProvider/plsql/ar_invoice_api_pub/APPS_XX_BPEL_CRE
ATE_SINGLE_INVOICE_AR_INVOICE_API_PUB-24CREATE_SIN.xsd"/>
   </schema>
...
```

In addition to message schema locations and schema elements that help to define the web messages, the *Types* element can also take a complex data type as input.

For example, the Responsibility, Responsibility Application, Security Group, NLS Language, and Organization ID complex types listed under the "SOAHeader" as shown below are used in passing values that would be used to set the applications context during service execution.

```
...
<schema xmlns="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified"
   targetNamespace="http://xmlns.oracle.
com/apps/ar/soaprovider/plsql/ar_invoice_api_pub/">
   <element name="SOAHeader">
     <complexType>
     <sequence>
      <element name="Responsibility" minOccurs="0" type="string"/>
          <element name="RespApplication" minOccurs="0" type="string"/>
          <element name="SecurityGroup" minOccurs="0" type="string"/>
          <element name="NLSLanguage" minOccurs="0" type="string"/>
                 <element name="Org_Id" minOccurs="0" type="string" />
      </sequence>
    </complexType>
    </element>
  </schema>
</types>
```

Message

The *Message* element defines the name of the message. It consists of one or more Part elements, which describe the content of a message using *Element* or *Type* attributes.

Parts are a flexible mechanism for describing the logical abstract content of a message. A binding may reference the name of a part in order to specify binding-specific information about the part.

The structure of this element can be like:

```
<definitions...>
 <message name="nmtoken"> *
   <part name="nmtoken" element="qname"? type="qname"? />
 </message>
</definitions>
```

A typical document-style web service could have a *header* and *body* part in the input message and output message as well. For example, the *Message* element for the "Invoice Creation" web service appears:

```
<message name="CREATE_INVOICE_Input_Msg">
   <part name="header" element="tns:SOAHeader"/>
   <part name="body" element="tns1:InputParameters"/>
</message>
<message name="CREATE_INVOICE_Output_Msg">
   <part name="body" element="tns1:OutputParameters"/>
</message>
<message name="CREATE_SINGLE_INVOICE_Input_Msg">
   <part name="header" element="tns:SOAHeader"/>
   <part name="body" element="tns2:InputParameters"/>
</message>
<message name="CREATE_SINGLE_INVOICE_Output_Msg">
   <part name="body" element="tns2:InputParameters"/>
</message>
```

Each message defined by the associated schema includes input message and output message parts. For example, the "Invoice Creation" web service has two functions:

- CREATE_INVOICE

  The input message of this function which has all its parameter is defined by **CREATE_INVOICE_Input_Msg**.

  The output message of this function which gives its result is defined by **CREATE_INVOICE_Output_Msg**.

  The schema of input and output messages is defined in `APPS_XX_BPEL_CREATE_INVOICE_AR_INVOICE_API_PUB-24CREATE_INV.xsd`.

- CREATE_SINGLE_INVOICE

  The input message of this function which has all its parameter is defined by **CREATE_SINGLE_INVOICE_Input_Msg**.

  The output message of this function which gives its result is defined by **CREATE_SINGLE_INVOICE_Output_Msg**.

  The schema of input and output messages is defined in `APPS_XX_BPEL_CREATE_SINGLE_INVOICE_AR_INVOICE_API_PUB-24CREATE_INV.xsd`

The value of body part of each message will be set as SOAP body; the value of header part will be set in the SOAP header which is required for the service authorization.

For more information, see Understanding Web Service Input Message Parts, page 12-22 .

PortType

The *portType* element combines multiple message elements to form a complete one-way or round-trip operation supported by a web service.

For example, a *portType* can combine one request (input message element) and one response (output message element) message into a single request/response operation for the synchronous request - response operation, the most commonly used in SOAP services.

If it is for one-way operation, then the operation would contain an Input element only.

The structure of this element can be like:

```
<wsdl:definitions...>
 <wsdl:portType name="nmtoken">*
     <operation name="nmtoken"/>
       <wsdl:input name="nmtoken"? message="qname">?
    </wsdl:input>
    <wsdl:output name="nmtoken"? message="qname">?
    </wsdl:output>
    <wsdl:fault name="nmtoken"? message="qname">?
    </wsdl:fault>
   </wsdl:operation>
 </wsdl:portype>
</wsdl:definitions>
```

> **Note:** An optional Fault element can be used for error handling in both request-response and solicit response Operation models. This feature is not supported in this release.

In this "Invoice Creation" service example, corresponding to above two functions, AR_INVOICE_API_PUB_PortType has the following two operations:

- CREATE_INVOICE
  - Input: CREATE_INVOICE_Input_Msg
  - Output: CREATE_INVOICE_Output_Msg
- CREATE_SINGLE_INVOICE
  - Input: CREATE_SINGLE_INVOICE_Input_Msg
  - Output: CREATE_SINGLE_INVOICE_Output_Msg

```
<portType name="AR_INVOICE_API_PUB_PortType">
     <operation name="CREATE_INVOICE">
       <input name="tns:CREATE_INVOICE_Input_Msg" />
    <output name="tns:CREATE_INVOICE_Output_Msg" />
   </operation>
   <operation name="CREATE_SINGLE_INVOICE">
       <input name="tns:CREATE_SINGLE_INVOICE_Input_Msg" />
    <output name="tns:CREATE_SINGLE_INVOICE_Output_Msg" />
   </operation>
</portype>
```

Binding

A *binding* defines message format and protocol details for operations and messages defined by a particular *portType*. It provides specific details on how a *portType* operation will actually be transmitted over the web. Bindings can be made available through multiple transports, including HTTP GET, HTTP POST, or SOAP.

A *port* defines an individual endpoint by specifying a single address for a binding.

The structure of this element can be like:

```
<wsdl:definitions...>
 <wsdl:binding name="nmtoken" type="qname">*
    <wsdl:operation name="nmtoken"/>
      <wsdl:input> ?
       </wsdl:input>
   <wsdl:output>?
   </wsdl:output>
   <wsdl:fault name="nmtoken"? message="qname">?
   </wsdl:fault>
  </wsdl:operation>
 </wsdl:binding>
</wsdl:definitions>
```

In the same example, the binding element as shown below describes the SOAP binding for PortType `AR_INVOICE_API_PUB_PortType`.

```
<binding name="AR_INVOICE_API_PUB_Binding" type="tns:
AR_INVOICE_API_PUB_PortType">
 <soap:binding style="document" transport="http://schemas.xmlsoap.
org/soap/http"/>
       <operation name="CREATE_INVOICE">
    <soap:operation
     soapAction="https://<hostname>:
<port>/webservices/SOAProvider/plsql/ar_invoice_api_pub/"/>
    <input>
     <soap:header message="tns:CREATE_INVOICE_Input_Msg" part="header"
use="literal" />
     <soap:body parts="body" use="literal" />
    </input>
    <output>
     <soap:body use="literal" />
     </output>
   </operation>
   <operation name="CREATE_SINGLE_INVOICE">
      <soap:operation
     soapAction="https://<hostname>:
<port>/webservices/SOAProvider/plsql/ar_invoice_api_pub/"/>
    <input>
     <soap:header message="tns:CREATE_SINGLE_INVOICE_Input_Msg" part="
header" use="literal" />
     <soap:body parts="body" use="literal" />
    </input>
    <output>
     <soap:body use="literal" />
     </output>
   </operation>
 </binding
```

The binding used is always document style, SOAP over `http` binding. It also defines the content of the SOAP header and SOAP body.

Note: Because it is a document-style service (`style="document"`), the request and response messages will consist of simply XML documents, instead of using the wrapper elements required for the remote procedure call (RPC-style) web service. The `transport` attribute indicates the transport of the SOAP messages is through SOAP HTTP.

Within each operation, the `soap:operation` element indicates the binding of a specific operation (such as CREATE_INVOICE) to a specific SOAP implementation. The `soapAction` attribute specifies that the SOAPAction HTTP header be used for identifying the service.

The `soap:header` element allows header to be defined that is transmitted inside the Header element of the SOAP Envelope. The `SOAHeader` comprises of Responsibility, RespApplication, SecurityGroup, NLSLanguage, and Org_Id complex types within the Types element.

The `soap:body` element enables you to specify the details of the input and output messages for a specific operation.

Service

The *service* element defines the web service, and typically consists of one or more *Port* elements. A port defines an individual endpoint by specifying a single address for a binding.

The service binding is commonly created using SOAP.

The structure of this element can be like:

```
<wsdl:definitions...>
 <wsdl:service name="nmtoken">*
     <wsdl:port name="nmtoken" binding="qname"> *
       </wsdl:port>
 </wsdl:service>
</wsdl:definitions>
```

In this example, the *Service* element `AR_INVOICE_API_PUB_Service` defines physical location of the service endpoint where the service is hosted for the portType `AR_INVOICE_API_PUB_PortType`.

```
<service name="AR_INVOICE_API_PUB_Service">
  <port name="AR_INVOICE_API_PUB_Port" binding="tns:
AR_INVOICE_API_PUB_Binding">
   <soap:address
   location="https://<hostname>:
<port>/webservices/SOAProvider/plsql/ar_invoice_api_pub/"/>
  </port>
 </service>
```

Reviewing WADL Element Details

If an interface is exposed as a REST service, you can view the corresponding WADL description in a separate window.

Take the same interface example PL/SQL API Invoice Creation (AR_INVOICE_API_PUB) explained earlier for WSDL description. This interface can also be exposed as a REST service. To view the associated WADL information, click **View WADL** link in the REST Web Service tab of the interface details page. The WADL document appears.

WADL Document Structure

WADL (Web Application Description Language) is designed to provide a machine processable description of HTTP-based Web applications.

The *application* element forms the root of a WADL description. It may contain the following elements:

- Grammars: This element serves as a container for definitions of data exchanged during execution of the protocol described by the WADL document.

- Resources: This element serves as a container for all the included child resource elements provided by the application.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<application xmlns:tns="http://xmlns.oracle.
com/apps/ar/soaprovider/plsql/rest/ar_invoice_api_pub/" xmlns="http:
//wadl.dev.java.net/2009/02"
xmlns:tns1="http://xmlns.oracle.com/apps/ar/rest/ar/create_invoice/"
name="AR_INVOICE_API_PUB"
targetNamespace="http://xmlns.oracle.
com/apps/ar/soaprovider/plsql/rest/ar_invoice_api_pub/">
   <grammars>
...
   </grammars>
   <resources base="http://<hostname>:<port>/webservices/rest/Invoice/">
  ...
 </resources
</application>
```

Grammars

This element acts as a container for definitions of data exchanged during execution of the protocol described by the WADL document. *Include* element is often referenced to allow the definitions of one or more data format descriptions to be included.

```
   <grammars>
    <include xmlns="http://www.w3.org/2001/XMLSchema" href="https:
//<hostname>:<port>/webservices/rest/Invoice/?XSD=CREATE_INVOICE.xsd" />

     <include xmlns="http://www.w3.org/2001/XMLSchema" href="https:
//<hostname>:<port>/webservices/rest/Invoice/?XSD=CREATE_SINGLE_INVOICE.
xsd" />
    </grammars>
```

Invoice highlighted here is the service alias name entered earlier prior to the service deployment. Once the service has been successfully deployed, the specified alias name (Invoice) becomes part of the service endpoint in the WADL and namespaces in the XSDs.

Resources

The *resources* element represents the resource information provided by the Web application. It includes a *base* attribute that provides the base URI for each included child resource identifier.

For example, each child *resource* element represents a specific service operation (such as create_invoice and create_single_invoice) contained in the selected interface.

```
<resources base="http://<hostname>:<port>/webservices/rest/Invoice/">
 <resource path="/create_invoice/">
  ...
 </resource>
 <resource path="/create_single_invoice/">
  ...
  </resource>
</resources>
```

Resource

A *resources* element may contain a set of *resource* elements; each resource element represents a REST service operation. In this example, create_invoice and create_single_invoice are included child resource element.

Each resource element can include the following child elements:

- *Method*: This element describes the input to and output from an HTTP protocol method that can be applied to the resource.

  POST is the only supported method for PL/SQL and Concurrent Program REST services; POST and GET are the supported methods for Java Bean Services, Application Module Services, and Business Service Object REST services. For Open Interfaces, the supported methods are determined based on the direction of the interface. For Open Interface Tables with `Inbound` direction, all four HTTP methods (GET, POST, PUT, and DELETE) are supported. Otherwise, only GET is supported for Open Interface Tables with `Outbound` direction and Open Interface Views.

- *Request*: This element describes the input to be included when applying an HTTP method to a resource.

  Element *mediaType* indicates the supported media type, such as XML and JSON, for the input parameters.

- *Response*: This element describes the output results from performing an HTTP method on a resource.

  The supported media types for the output results are XML and JSON.

```
 <resources base="http://<hostname>:<port>/webservices/rest/Invoice/">
  <resource path="/create_invoice/">
   <method id="CREATE_INVOICE" name="POST">
    <request>
     <representation mediaType="application/xml" type="tns1:
InputParameters" />
     <representation mediaType="application/json" type="tns1:
InputParameters" />
    </request>
    <response>
     <representation mediaType="application/xml" type="tns1:
OutputParameters" />
     <representation mediaType="application/json" type="tns1:
OutputParameters" />
    </response>
   </method>
  </resource>
  <resource path="/create_single_invoice/">
   <method id="CREATE_SINGLE_INVOICE" name="POST">
    <request>
     <representation mediaType="application/xml" type="tns2:
InputParameters" />
     <representation mediaType="application/json" type="tns2:
InputParameters" />
    </request>
    <response>
     <representation mediaType="application/xml" type="tns2:
OutputParameters" />
     <representation mediaType="application/json" type="tns2:
OutputParameters" />
    </response>
   </method>
  </resource>
 </resources>
```

In this example, input request and output response messages are all supported with XML and JSON formats when applying the HTTP method POST for each resource element 'create_invoice' and 'create_single_invoice'.

If the deployed REST service is an interface type of Java Bean Services or Application Module Services, then both GET and POST can be shown as the supported methods in the REST service operation. For example, the following WADL description shows two of many methods contained in the Rest Service Locator interface. addGrant is implemented with POST HTTP method, and getOperations is assisted with GET HTTP method.

```
  <resources base="http://<hostname>:<port>/webservices/rest/locator/">
  <resource path="/addGrant/">
   <method id="addGrant" name="POST">
    <request>
     <representation mediaType="application/xml" type="tns1:
addGrant_InputParameters" />
     <representation mediaType="application/json" type="tns1:
addGrant_InputParameters" />
    </request>
   <response>
     <representation mediaType="application/xml" type="tns1:
addGrant_OutputParameters" />
     <representation mediaType="application/json" type="tns1:
addGrant_OutputParameters" />
    </response>
   </method>
  </resource>
...
  <resource path="/getOperations/{irepClassName}/">
    <param name="irepClassName" style="template" required="true" type="
xsd:string"/>
   <method id="getOperations" name="GET">
    <request>
     <param name="ctx_responsibility" type="xsd:string" style="query"
required="false" />
              <param name="ctx_respapplication" type="xsd:string"
style="query" required="false" />
     <param name="ctx_securitygroup" type="xsd:string" style="query"
required="false" />
     <param name="ctx_nlslanguage" type="xsd:string" style="query"
required="false" />
     <param name="ctx_orgid" type="xsd:int" style="query" required="
false" />
    </request>
      <response>
     <representation mediaType="application/xml" type="tns3:
getOperations_OutputParameters" />
     <representation mediaType="application/json" type="tns3:
getOperations_OutputParameters" />
    </response>
   </method>
  </resource>
</resources>
```

The GET method of the above example contains the path variable and application context information if required for the service execution:

> **Note:** Path variables and context parameters are applicable only for the HTTP GET method here, as well as the DELETE method as shown in the following example for Open Interface Table.

- {irepClassName} is a **path variable** for "getOperations" operation as specified below:

  `<resource path="/getOperations/{irepClassName}/">`

  Path variable is defined using the `<param>` tag after the `<resource>` tag and before the `<method>` tag. A service client will replace the path variable with actual value at run time when the HTTP GET method is invoked.

  For example, if the value of the `irepClassName` is "PLSQL:FND_PROFILE", then

"PLSQL:FND_PROFILE" should be passed in HTTP URL for this request. The URL on which the HTTP GET will be performed is: `http://<hostname>:<port>/webservices/rest/locator/getOperations/PLSQL:FND_PROFILE`

Path variables are identified by inline annotation `@rep:key_param` in Java API source file. For annotation guidelines on Java Bean Services, see Annotations for Java Bean Services, page A-6. For annotation guidelines on Application Module Services, see Annotations for Application Module Services, page A-6.

- Context parameters are predefined parameters required for initializing Oracle E-Business Suite application context. These parameters including `ctx_responsibility, ctx_respapplication, ctx_securitygroup, ctx_nlslanguage,` and `ctx_orgid` are applicable only for the HTTP GET method and DELETE method (as shown in the following example for Open Interface Table). These parameters are passed as query strings along with the request payload query strings in the RESTHeader element.

Control parameters are predefined query parameters that may be used for a resource or collection resource. For example, use `offset` and `limit` parameters to limit the number of records returned and for pagination.

`offset=<number>&limit=<number>`

- `offset=0&limit5`

  This returns the first 5 records of response after record 0. That is the 1st, 2nd, 3rd, 4th, and 5th records.

- `offset=2&limit4`

  This returns the first 4 records of response after record 2. That is the 3rd, 4th, 5th and 6th records.

- `http://<hostname>:<port>/webservices/rest/empinfo/getAllReports/?offset=0&limit=5`

  This returns the first 5 employees reporting to a logged in user in hierarchical order.

If the deployed REST service is an open interface table with Inbound direction, then the service operation table displays all four HTTP methods. In the following WADL example for the `AR Autoinvoice` (associated concurrent program internal name `RAXMTR`) open interface table, the `RA_INTERFACE_LINES_ALL` operation is implemented with all four HTTP methods, and the associated concurrent program `SUBMIT_CP_RAXMTR` is implemented with POST method.

```
 <resources base="http://<hostname>:<port>/webservices/rest/autoinvoice
/">
  <resource path="RA_INTERFACE_LINES_ALL/">
 <method id="RA_INTERFACE_LINES_ALL_get" name="GET">
     <request>
    <param name="ctx_responsibility" type="xsd:string" style="query"
required="false"/>
          <param name="ctx_respapplication" type="xsd:string" style="
query" required="false" />
    <param name="ctx_securitygroup" type="xsd:string" style="query"
required="false" />
    <param name="ctx_nlslanguage" type="xsd:string" style="query"
required="false" />
    <param name="ctx_orgid" type="xsd:int" style="query" required="
false" />
    <param name="select" type="xsd:string" style="query" required="
false"/>
    <param name="filter" type="xsd:string" style="query" required="
false"/>
    <param name="sort" type="xsd:string" style="query" required="false"
/>
    <param name="offset" type="xsd:string" style="query" required="
false"/>
    <param name="limit" type="xsd:string" style="query" required="false"
/>
   </request>
   <response>
    <representation mediaType="application/xml" type="tns1:
RA_INTERFACE_LINES_ALL_Output"/>
    <representation mediaType="application/json" type="tns1:
RA_INTERFACE_LINES_ALL_Output" />
    <representation mediaType="text/csv" type="tns1:
RA_INTERFACE_LINES_ALL_Output"/>
   </response>
 </method>
 <method id="RA_INTERFACE_LINES_ALL_post" name="POST">
     <request>
    <representation mediaType="application/xml" type="tns1:
RA_INTERFACE_LINES_ALL_Input"/>
    <representation mediaType="application/json" type="tns1:
RA_INTERFACE_LINES_ALL_Input" />
    <representation mediaType="text/csv" type="tns1:
RA_INTERFACE_LINES_ALL_Input"/>
   </request>
   <response>
    <representation mediaType="application/xml" type="tns1:
RA_INTERFACE_LINES_ALL_Output"/>
    <representation mediaType="application/json" type="tns1:
RA_INTERFACE_LINES_ALL_Output" />
    <representation mediaType="text/csv" type="tns1:
RA_INTERFACE_LINES_ALL_Output"/>
   </response>
 </method>
 <method id="RA_INTERFACE_LINES_ALL_put" name="PUT">
     <request>
    <representation mediaType="application/xml" type="tns1:
RA_INTERFACE_LINES_ALL_Input"/>
    <representation mediaType="application/json" type="tns1:
RA_INTERFACE_LINES_ALL_Input" />
    <representation mediaType="text/csv" type="tns1:
RA_INTERFACE_LINES_ALL_Input"/>
   </request>
   <response>
    <representation mediaType="application/xml" type="tns1:
RA_INTERFACE_LINES_ALL_Output"/>
    <representation mediaType="application/json" type="tns1:
```

```
RA_INTERFACE_LINES_ALL_Output" />
    <representation mediaType="text/csv" type="tns1:
RA_INTERFACE_LINES_ALL_Output"/>
   </response>
 </method>
 <method id="RA_INTERFACE_LINES_ALL_delete" name="DELETE">
     <request>
    <param name="ctx_responsibility" type="xsd:string" style="query"
required="false"/>
          <param name="ctx_respapplication" type="xsd:string" style="
query" required="false" />
    <param name="ctx_securitygroup" type="xsd:string" style="query"
required="false" />
    <param name="ctx_nlslanguage" type="xsd:string" style="query"
required="false" />
    <param name="ctx_orgid" type="xsd:int" style="query" required="
false" />
    <param name="filter" type="xsd:string" style="query" required="
false"/>
   </request>
   <response>
    <representation mediaType="application/xml" type="tns1:
RA_INTERFACE_LINES_ALL_Output"/>
    <representation mediaType="application/json" type="tns1:
RA_INTERFACE_LINES_ALL_Output" />
    <representation mediaType="text/csv" type="tns1:
RA_INTERFACE_LINES_ALL_Output"/>
   </response>
 </method>
</resource>
<resource path="SUBMIT_CP_RAXMTR/">
 <method id="SUBMIT_CP_RAXMTR_post" name="POST">
      <request>
    <representation mediaType="application/xml" type="tns1:
SUBMIT_CP_RAXMTR_Input"/>
    <representation mediaType="application/json" type="tns1:
SUBMIT_CP_RAXMTR_Input" />
    <representation mediaType="text/csv" type="tns1:
SUBMIT_CP_RAXMTR_Input"/>
   </request>
   <response>
    <representation mediaType="application/xml" type="tns1:
SUBMIT_CP_RAXMTR_Output"/>
    <representation mediaType="application/json" type="tns1:
SUBMIT_CP_RAXMTR_Output" />
    <representation mediaType="text/csv" type="tns1:
SUBMIT_CP_RAXMTR_Output"/>
   </response>
 </resource>
```

- Each open interface table name contained in the selected open interface "AR Autoinvoice" is displayed in one resource entry (`<resource path>`) with the selected HTTP method(s). In this example, table name `RA_INTERFACE_LINES_ALL` with the four selected methods (`GET`, `POST`, `PUT`, and `DELETE`) is contained in one resource entry, and the associated concurrent program `SUBMIT_CP_RAXMTR` with `POST` is contained in another resource entry.

- The WADL description for Open Interface View contains only one resource entry with GET method only. There is no concurrent program `SUBMIT_CP_<internal name of the concurrent program>` associated with the Open Interface View.

Understanding SOAP Messages

SOAP (Simple Object Access Protocol) is a lightweight, XML-based protocol specification for exchanging structured information in the implementation of Web services in computer networks. For example, Web service provider receives SOAP requests from Web service clients to invoke Web services and also sends the corresponding SOAP responses out to the clients.

To support all integration interface types and services in Oracle E-Business Suite Integrated SOA Gateway, all SOAP messages are authenticated, authorized, and service enabled through SOA Provider except for Business Service Object services and generic XML Gateway messages that are enabled through Web Service Provider.

SOAP Message Structure

SOAP is an XML-based protocol and acts as a building block for Web service communication. SOAP messages are contained in one of the SOAP components called *Envelope*. The SOAP envelop defines an overall framework for describing what is in a message; who should deal with it, and whether it is optional or mandatory. It consists of the following elements:

- Header (Optional)

  An envelope element can optionally have a Header element. If an envelope contains a Header element, it must contain no more than one, and it must appear as the first child of the envelope. The first level child elements of the Header element are called Header Blocks.

  Header blocks can be used in the following mechanisms:

  - It provides a mechanism for attaching security related information targeted at a specific recipient.

    For more information, see SOAP Security Header, page 2-25.

  - It can be used to set applications context values required for services.

    For more information, see SOAP Header for Applications Context, page 2-29.

  - It can be used to populate mandatory header variables for XML Gateway inbound transactions to be completed successfully.

    For more information, see SOAP Header for XML Gateway Messages, page 2-32

- Body

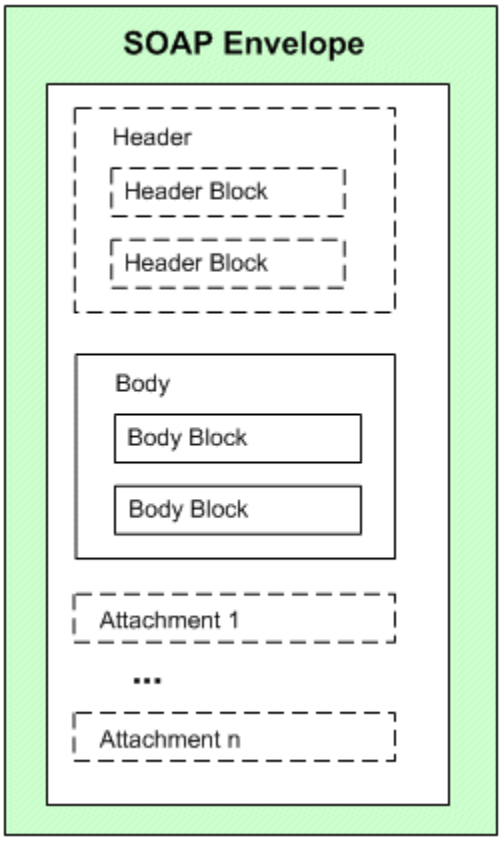
  Every envelope element must contain exactly one Body element that holds the message. Immediate child elements of the Body element are called Body Blocks or Parts.

- Attachment (Optional)

  A SOAP message can carry multiple attachments and these attachments can be of any type including text, binary, image, and so on.

The following diagram depicts the structure of a SOAP message.

SOAP Message Structure



A skeleton of a SOAP message can be like:

```
<xml version="1.0">
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header>
...
</soap:Header>

<soap:Body>
...
   <soap:Fault>
    ...
   </soap:Fault>
</soap:Body>

</soap:Envelope>
```

SOAP Security Header

When a SOAP request message is received through SOA Provider, the SOAP message is passed on to OC4J Web Service Framework for authentication. The framework authenticates the SOAP message based on the specified authentication type(s) during the service deployment. The identified authentication information is embedded in the `wsse:security` Web Security headers.

UsernameToken-based SOAP Security Header

A UsernameToken-based SOAP header should include the following `wsse:security` section:

```
<soapenv:Header>
<http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd"
soapenv:mustUnderstand="1">
 <wsse:UsernameToken>
  <wsse:Username>Username</wsse:Username>
  <wsse:Password>Password</<wsse:Password>
 </wsse:UsernameToken>
</wsse:Security>

</soapenv:Header>
```

> **Note:** When a `<wsse:security>` header includes a `mustUnderstand="1"` attribute, then the receiver must generate a `fault` if it is unable to interpret or process security tokens contained the `<wsse:security>` header block according to the corresponding WS SOAP message security token profiles.
>
> See A Sample Fault SOAP Response for Business Service Object, page 2-41.

A typical WS-Security header in a SOAP Request can be like:

```
<soapenv:Header>
<http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd"
soapenv:mustUnderstand="1">
 <wsse:UsernameToken>
  <wsse:Username>myUser</wsse:Username>
  <wsse:Password
  Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
username-token-profile-1.0#PasswordText">password</wsse:Password>
 </wsse:UsernameToken>
</wsse:Security>

</soapenv:Header>
```

The UsernameToken based security mechanism includes UsernameToken profile which provides username and password information in the Web service security header. Username is a clear text; password is the most sensitive part of the UsernameToken profile. In this security model, the supported password type is plain text password (or PasswordText).

The username/password in SOAP Header of a SOAP message will be passed for Web service authentication. The username/password discussed here in `wsse:security` is the Oracle E-Business Suite username/password (or the username/password created through the Users window in defining an application user).

Passing security header elements along with the SOAP request is essential to the success of invoking Oracle E-Business Suite Web services through SOA Provider or Web Service Provider.

If these security header values are not passed, the Web service will not be authenticated and the execution of the service will be failed.

Detailed instructions on how to pass the security header along with the SOAP request when invoking an Oracle E-Business Suite Web service from a BPEL process, see Passing Values to Security Headers, page 3-10.

SAML Token-based SOAP Security Header

Security Assertion Markup Language (SAML) is an XML-based standard for exchanging authentication and authorization data between security domains, that is, between an identity provider and a service provider.

When a Web application invokes a service that uses SAML as its authentication mechanism, this SOAP request message containing or referencing SAML assertions is received through SOA Provider and passed on to OC4J Web Service Framework for authentication. The framework authenticates the SOAP message based on the `wsse:security` Web Security headers. As part of the validation and processing of the assertions, the receiver or authentication framework must establish the relationship between the subject, claims of the referenced SAML assertions, and the entity providing the evidence to satisfy the confirmation method defined for the statements.

A trusted entity uses the sender-vouches confirmation method to ensure that it is acting on behalf of the subject of SAML statements attributed with a sender-vouches `SubjectConfirmation` element.

The following SOAP example describes a trusted entity uses the sender-vouches subject confirmation method with an associated `<ds:Signature>` element to establish its identity and to assert that it has sent the message body on behalf of the subject(s):

```
<soapenv:Envelope
xmlns:fnd="http://xmlns.oracle.
com/apps/fnd/soaprovider/plsql/fnd_user_pkg/" xmlns:soapenv="http:
//schemas.xmlsoap.org/soap/envelope/">
 <soapenv:Header>
  <wsse:Security xmlns:wsse="http://docs.oasis-open.
org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
    <ds:Signature Id="Signature-xxxxxxxx" xmlns:ds="http://www.w3.
org/2000/09/xmldsig#">
    <ds:SignedInfo>
    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-
exc-c14n#"/>
    <ds:SignatureMethod Algorithm="http://www.w3.
org/2000/09/xmldsig#rsa-sha1"/>
    <ds:Reference URI="#id-xxxxxxxx">
    <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
    </ds:Transforms>
    <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"
/>
    <ds:DigestValue>xxx/xxxxxxxxxxxxxxxxxxxx/xx</ds:DigestValue>
    </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue>

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx/xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxx/
xxxxxxxxxxxxxxxxx
   </ds:SignatureValue>
   <ds:KeyInfo Id="KeyId-xxxxxxx">
 <wsse:SecurityTokenReference wsu:Id="STRId-xxxxxxx" xmlns:wsu="http:
//docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd"><wsse:KeyIdentifier
   EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-soap-message-security-1.0#Base64Binary"
   ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
x509-token-profile-1.0#X509SubjectKeyIdentifier"
>ADoNKKuduSTKTwi7jqEzCxwD7JU=</wsse:KeyIdentifier></wsse:
SecurityTokenReference>
  </ds:KeyInfo></ds:Signature>
  <Assertion AssertionID="xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
IssueInstant="2010-02-27T17:26:21.241Z" Issuer="www.oracle.com"
MajorVersion="1" MinorVersion="1" xmlns="urn:oasis:names:tc:SAML:1.0:
assertion" xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion" xmlns:
samlp="urn:oasis:names:tc:SAML:1.0:protocol" xmlns:xsd="http://www.w3.
org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"><Conditions NotBefore="2010-02-27T17:26:21.241Z"
NotOnOrAfter="2011-02-27T17:26:21.241Z"/>
    <AuthenticationStatement AuthenticationInstant="2010-02-27T17:26:
21.241Z" AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:password">
       <Subject>
         <NameIdentifier Format="urn:oasis:names:tc:SAML:1.1:nameid-
format:unspecified"
          NameQualifier="notRelevant">SYSADMIN</NameQualifier>
         <SubjectConfirmation>
    <ConfirmationMethod>urn:oasis:names:tc:SAML:1.0:cm:sender-
vouches</ConfirmationMethod>
         </SubjectConfirmation>
       </Subject>
   </AuthenticationStatement>
  </Assertion>
 </wsse:Security>


     <fnd:SOAHeader>
```

```
<!--Optional:-->
        <fnd:Responsibility>UMX</fnd:Responsibility>
        <!--Optional:-->
        <fnd:RespApplication>FND</fnd:RespApplication>

     </fnd:SOAHeader>
   </soapenv:Header>

  <soapenv:Body wsu:Id="id-xxxxxxxx" xmlns:wsu="http://docs.oasis-open.
org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
    <tes:InputParameters xmlns:tes="http://xmlns.oracle.
com/apps/fnd/soaprovider/plsql/fnd_user_pkg/testusername/">
       <!--Optional:-->
       <tes:X_USER_NAME>username</tes:X_USER_NAME>
    </tes:InputParameters>
 </soapenv:Body>
</soapenv:Envelope>
```

> **Note:** SAML Token based security can be used to authenticate users in both Single Sign-On (SSO) and non-SSO enabled environments. The format of the `NameIdentifier` in the SAML assertion indicates if the user has been authenticated against LDAP (SSO user) or Oracle E-Business Suite `FND_USER` table (for non-SSO user).
>
> The SAML assertion in the above SOAP message is for non-SSO enabled environment. If the username in the `NameIdentifier` tag is of the form of LDAP DN as shown below, then the username is verified in the registered OID for SSO user.
>
> ```
> <NameIdentifier Format="urn:oasis:names:tc:SAML:1.1:
> nameid-format:unspecified"
>      NameQualifier="notRelevant"
> >orclApplicationCommonName=PROD1,cn=EBusiness,cn=Products,
> cn=OracleContext,dc=us,dc=oracle,dc=com</NameIdentifier>
> ```

For more information about SAML Token sender-vouches based security, see SAML Sender-Vouches Token Based Security, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

SOAP Header for Applications Context

Applications context contains many crucial elements that are used in passing values that may be required in proper functioning of Oracle E-Business Suite Web services. For example, the context header information is required for an API transaction or a concurrent program in order for an Oracle E-Business Suite user that has sufficient privileges to run the program.

Applications Context in `SOAHeader` Part of a SOAP Request

These context header elements defined in `SOAHeader` part of a SOAP request for PL/SQL and Concurrent Program services are:

- Responsibility

  It is the Oracle E-Business Suite application responsibility information. It accepts

responsibility_key (such as SYSTEM_ADMINISTRATOR) as its value.

- RespApplication

  It is the responsibility application short name information. It accepts Application Short Name (such as FND) as its value.

- SecurityGroup

  It accepts Security Group Key (such as STANDARD) as its value.

- NLSLanguage (optional)

  It is an optional parameter to be passed in SOAHeader part of a SOAP request for PL/SQL and Concurrent Program services.

  If the NLS Language element is specified, SOAP requests can be consumed in the language passed. All corresponding SOAP responses and error messages can also be returned in the same language. If no language is identified, then the default language of the user will be used.

- Org_Id (optional for PL/SQL and Concurrent Program services)

  It is an optional parameter to be passed in SOAHeader part of a SOAP request for PL/SQL and Concurrent Program services. If a service execution is dependent on any particular organization, then you must pass the Org_Id element of that SOAP request.

The following SOAP message shows the SOAHeader part highlighted in bold text:

```
<soapenv:Header>
<http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd"
soapenv:mustUnderstand="1">
 <wsse:UsernameToken>
  <wsse:Username>myUser</wsse:Username>
  <wsse:Password
  Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
username-token-profile-1.0#PasswordText">password</wsse:Password>
 </wsse:UsernameToken>
</wsse:Security><ozf:SOAHeader>
 <ozf:Responsibility>OZF_USER</ozf:Responsibility>
 <ozf:RespApplication>OZF</ozf:RespApplication>
 <ozf:SecurityGroup>STANDARD</ozf:SecurityGroup>
 <ozf:NLSLanguage>AMERICAN</ozf:NLSLanguage>
 <ozf:Org_Id>204</ozf:Org_Id>
</ozf:SOAHeader>
</soapenv:Header>
```

Applications Context in ServiceBean_Header Part of a SOAP Request

These context header elements defined in ServiceBean_Header part of a SOAP request for Business Service Object services are:

- RESPONSIBILITY_NAME

  It is the Oracle E-Business Suite application responsibility information. It can accept

both the name (`Responsibility_Name`, such as `System Administrator`) and the key (in the format of `{key}responsibility_key`, such as `{key} SYSTEM_ADMINISTRATOR`) as its values.

- RESPONSIBILITY_APPL_NAME

  It is the responsibility application short name information. It accepts Application Short Name (such as `FND`) as its value.

- SECURITY_GROUP_NAME

  It accepts Security Group Key (such as `STANDARD`) as its value.

- NLSLanguage (optional)

  It is an optional parameter to be passed in `ServiceBean_Header` part of a SOAP request for Business Service Object service.

  If the NLS Language element is specified (such as `AMERICAN`), the SOAP request can be consumed in the language passed. All corresponding SOAP responses and error messages can also be returned in the same language. If no language is identified, then the default language of the user will be used.

- Org_Id (optional)

  It is an optional parameter to be passed in `ServiceBean_Header` part of a SOAP request for Business Service Object service.

  If a service execution is dependent on any particular organization, then you must pass the Org_Id element of that SOAP request.

The following SOAP request example includes the `ServiceBean_Header` part highlighted in bold text for business service object:

```
<soapenv:Envelope xmlns:ser="http://xmlns.oracle.
com/apps/fnd/ServiceBean"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ws="
http://xmlns.oracle.com/apps/fnd/rep/ws">
  <soapenv:Header>
   <wsse:Security soapenv:mustUnderstand="1" xmlns:wsse="http://docs.
oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
   <wsse:UsernameToken wsu:Id="UsernameToken-22948433" xmlns:wsu="http:
//docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd">
      <wsse:Username>sysadmin</wsse:Username>
      <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-username-token-profile-1.0#PasswordText">password</wsse:
Password>
    </wsse:UsernameToken>
   </wsse:Security><ser:ServiceBean_Header>
    <ser:RESPONSIBILITY_NAME>System Administrator</ser:
RESPONSIBILITY_NAME>
    <ser:RESPONSIBILITY_APPL_NAME>sysadmin</ser:
RESPONSIBILITY_APPL_NAME>
    <ser:SECURITY_GROUP_NAME>standard</ser:SECURITY_GROUP_NAME>
    <ser:NLS_LANGUAGE>american</ser:NLS_LANGUAGE>
    <ser:ORG_ID>202</ser:ORG_ID>
  </ser:ServiceBean_Header>
 </soapenv:Header>
 <soapenv:Body>
   <ws:IntegrationRepositoryService_GetInterfaceByType>
     <interfaceType>XMLGATEWAY</interfaceType>
   </ws:IntegrationRepositoryService_GetInterfaceByType>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP Header for XML Gateway Messages

The SOAP header part can also be used to populate header variables for XML Gateway inbound transactions to be completed successfully. These XML Gateway header parameters defined in the `SOAHeader` (through SOA Provider) or `XMLGateway_Header` (through Web Service Provider) part of a SOAP Request are described in the following table:

The following code snippet shows the `SOAHeader` part of a SOAP request for an XML Gateway inbound message through SOA Provider:

```
<soapenv: Envelope xmlns:ecx="http://xmlns.oracle.
com/apps/ecx/soaprovider/xmlgateway/ecx__cbodi/"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:sys="http://xmlns.oracle.com/xdb/SYSTEM">
  <soapenv:Header>
  <wsse:Security soapenv:mustUnderstand="1"
     xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-wssecurity-secext-1.0.xsd">
      <wsse:UsernameToken wsu:Id="UsernameToken-10586449"
      xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-wssecurity-utility-1.0.xsd">
     <wsse:Username>SYSADMIN</wsse:Username>
         <wsse:Password Type="http://docs.oasis-open.
org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0
#PasswordText">password</wsse:Password>
  </wsse:UsernameToken>
    </wsse:Security>
    <ecx:SOAHeader>
     <sys:ECXMSG>
       <MESSAGE_TYPE></MESSAGE_TYPE>
       <MESSAGE_STANDARD></MESSAGE_STANDARD>
       <TRANSACTION_TYPE></TRANSACTION_TYPE>
       <TRANSACTION_SUBTYPE></TRANSACTION_SUBTYPE>
       <DOCUMENT_NUMBER></DOCUMENT_NUMBER>
   <PARTYID></PARTYID>
      <PARTY_SITE_ID></PARTY_SITE_ID>
   <PARTY_TYPE></PARTY_TYPE>
   <PROTOCOL_TYPE></PROTOCOL_TYPE>
   <PROTOCOL_ADDRESS></PROTOCOL_ADDRESS>
   <USERNAME></USERNAME>
   <PASSWORD></PASSWORD>
   <ATTRIBUTE1></ATTRIBUTE1>
   <ATTRIBUTE2></ATTRIBUTE2>
   <ATTRIBUTE3></ATTRIBUTE3>
   <ATTRIBUTE4></ATTRIBUTE4>
   <ATTRIBUTE5></ATTRIBUTE5>
  </sys:ECXMSG>
   </ecx:SOAHeader>
 </soapenv:Header>
```

The following table describes the XML Gateway header information in SOAHeader part of a SOAP request:

XMLGateway Header Information in SOAHeader Part of a SOAP Request

Attribute	Description
MESSAGE_TYPE	Payload message format. This defaults to `XML`. Oracle XML Gateway currently supports only `XML`.
MESSAGE_STANDARD	Message format standard as displayed in the Define Transactions form and entered in the Define XML Standards form. This defaults to `OAG`. The message standard entered for an inbound XML document must be the same as the message standard in the trading partner setup.

Attribute	Description
TRANSACTION_TYPE	External Transaction Type for the business document from the Trading Partner table. The transaction type for an inbound XML document must be the same as the transaction type defined in the Trading Partner form.
TRANSACTION_SUBTYPE	External Transaction Subtype for the business document from the Trading Partner table. The transaction subtype for an inbound XML document must be the same as the transaction subtype defined in the Trading Partner form.
DOCUMENT_NUMBER	The document identifier used to identify the transaction, such as a purchase order or invoice number. This field is not used by the XML Gateway, but it may be passed on inbound messages.
PROTOCOL_TYPE	Transmission Protocol is defined in the Trading Partner table.
PROTOCOL_ADDRESS	Transmission address is defined in the Trading Partner table.
USERNAME	USERNAME is defined in the Trading Partner table.
PASSWORD	The password associated with the USERNAME is defined in the Trading Partner table.
PARTY_SITE_ID	The party site identifier for an inbound XML document must be the same as the Source Trading Partner location defined in the Trading Partner form.
ATTRIBUTE1	This parameter may be defined by the base application.
ATTRIBUTE2	This parameter may be defined by the base application.

Attribute	Description
ATTRIBUTE3	For outbound messages, this field has the value from the Destination Trading Partner Location Code in the Trading Partner table. For inbound messages, the presence of this value generates another XML message that is sent to the trading partner identified in the Destination Trading Partner Location Code in the Trading Partner table. This value must be recognized by the hub to forward the XML message to the final recipient of the XML Message. **Note:** For more information, see *Destination Trading Partner Location Code* in the *Oracle XML Gateway User's Guide*.
ATTRIBUTE4	This parameter may be defined by the base application.
ATTRIBUTE5	This parameter may be defined by the base application.

- The Username and Password in `SOAHeader` here is the username and password associated with trading partner setup.

  The Username and Password in `<wsse:Security>` discussed earlier is the Oracle E-Business Suite username/password (or the username/password created through the Users window in defining an application user).

- The PARTYID and PARTY_TYPE parameters are note used.

The following code snippet shows the `XMLGateway_Header` part of a SOAP request through Web Service Provider:

```
<soap:Envelope>
  <soap:Header>
...
   <ns1:XMLGateway_Header
      xmlns:ns1="http://xmlns.oracle.com/apps/fnd/XMLGateway
      soapenv:mustUnderstand="0">
     <ns1:MESSAGE_TYPE>XML</ns1:MESSAGE_TYPE>
     <ns1:MESSAGE_STANDARD>OAG</ns1:MESSAGE_STANDARD>
     <ns1:TRANSACTION_TYPE>PO</ns1:TRANSACTION_TYPE>
     <ns1:TRANSACTION_SUBTYPE>PROCESS</ns1:TRANSACTION_SUBTYPE>
     <ns1:DOCUMENT_NUMBER>123</ns1:DOCUMENT_NUMBER>
     <ns1:PARTY_SITE_ID>4444</ns1:PARTY_SITE_ID>
   </ns1:XMLGateway_Header>
 </soap:Header>
...
</soap:Envelope>
```

The following table describes the XML Gateway header information in XMLGateway_Header part of a SOAP request:

XMLGateway_Header Part of a SOAP Request

Parameter Name	Description
MESSAGE_TYPE	Payload message format. This defaults to `XML`. Oracle XML Gateway currently supports only `XML`.
MESSAGE_STANDARD	Message format standard as displayed in the Define Transactions form and entered in the Define XML Standards form. This defaults to `OAG`. The message standard entered for an inbound XML document must be the same as the message standard in the trading partner setup.
TRANSACTION_TYPE	External Transaction Type for the business document from the Trading Partner table. The transaction type for an inbound XML document must be the same as the transaction type defined in the Trading Partner form.
TRANSACTION_SUBTYPE	External Transaction Subtype for the business document from the Trading Partner table. The transaction subtype for an inbound XML document must be the same as the transaction subtype defined in the Trading Partner form.
DOCUMENT_NUMBER	The document identifier used to identify the transaction, such as a purchase order or invoice number. This parameter is not used by the XML Gateway, but it may be passed on inbound messages.

Parameter Name	Description
PARTY_SITE_ID	The party site identifier for an inbound XML document must be the same as the Source Trading Partner location defined in the Trading Partner form.

Examples of SOAP Messages Through SOA Provider

To better understand SOAP request and response messages received through SOA Provider, the following sample SOAP messages are described in this section:

- A Sample SOAP Request, page 2-37
- A Sample SOAP Response, page 2-39
- A Sample Fault SOAP Response, page 2-39

A Sample SOAP Request

The following example shows a SOAP request for a PL/SQL service:

```
<soapenv:Envelope xmlns:ser="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd"
xmlns:ozf="http://xmlns.oracle.
com/apps/ozf/soaprovider/plsql/ozf_sd_request_pub/"
xmlns:cre="http://xmlns.oracle.
com/apps/ozf/soaprovider/plsql/ozf_sd_request_pub/create_sd_request/">
  <soapenv:Header>
 <wsse:Security soapenv:mustUnderstand="1">
    <wsse:UsernameToken>
  <wsse:Username>trademgr</wsse:Username>
  <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-username-token-profile-1.0#PasswordText">password</wsse:
Password>
  </wsse:UsernameToken>
   </wsse:Security>
   <ozf:SOAHeader>
     <ozf:Responsibility>OZF_USER</ozf:Responsibility>
     <ozf:RespApplication>OZF</ozf:RespApplication>
     <ozf:SecurityGroupE>STANDARD</ozf:SecurityGroup>
     <ozf:NLSLanguage>AMERICAN</ozf:NLSLanguage>
     <ozf:Org_Id>204</ozf:Org_Id>
   </ozf:SOAHeader>
 </soapenv:Header>
 <soapenv:Body>
   <cre:InputParameters>
  <cre:P_API_VERSION_NUMBER>1.0</cre:P_API_VERSION_NUMBER>
  <cre:P_INIT_MSG_LIST>T</cre:P_INIT_MSG_LIST>
     <cre:P_COMMIT>F</cre:P_COMMIT>
  <cre:P_VALIDATION_LEVEL>100</cre:P_VALIDATION_LEVEL>
      <cre:P_SDR_HDR_REC>
        <cre:REQUEST_NUMBER>SDR-CREATE-A1</cre:REQUEST_NUMBER>
        <cre:REQUEST_START_DATE>2008-08-18T12:00:00</cre:
REQUEST_START_DATE>
      <cre:REQUEST_END_DATE>2008-10-18T12:00:00</cre:REQUEST_END_DATE>>
        <cre:USER_STATUS_ID>1701</cre:USER_STATUS_ID>
        <cre:REQUEST_OUTCOME>IN_PROGRESS</cre:REQUEST_OUTCOME>
        <cre:REQUEST_CURRENCY_CODE>USD</cre:REQUEST_CURRENCY_CODE>
        <cre:SUPPLIER_ID>601</cre:SUPPLIER_ID>
        <cre:SUPPLIER_SITE_ID>1415</cre:SUPPLIER_SITE_ID>
        <cre:REQUESTOR_ID>100001499</cre:REQUESTOR_ID>
    <cre:ASSIGNEE_RESOURCE_ID>100001499</cre:ASSIGNEE_RESOURCE_ID>
    <cre:ORG_ID>204</cre:ORG_ID>
    <cre:ACCRUAL_TYPE>SUPPLIER</cre:ACCRUAL_TYPE>
    <cre:REQUEST_DESCRIPTION>Create</cre:REQUEST_DESCRIPTION>

       <cre:SUPPLIER_CONTACT_EMAIL_ADDRESS>sdr.supplier@example.
com</cre:SUPPLIER_CONTACT_EMAIL_ADDRESS>
       <cre:SUPPLIER_CONTACT_PHONE_NUMBER>2255</cre:
SUPPLIER_CONTACT_PHONE_NUMBER>
     <cre:REQUEST_TYPE_SETUP_ID>400</cre:REQUEST_TYPE_SETUP_ID>
       <cre:REQUEST_BASIS>Y</cre:REQUEST_BASIS>
       <cre:USER_ID>1002795</cre:USER_ID>
   </cre:P_SDR_HDR_REC>
   <cre:P_SDR_LINES_TBL>
     <cre:P_SDR_LINES_TBL_ITEM>
       <cre:PRODUCT_CONTEXT>PRODUCT</cre:PRODUCT_CONTEXT>
       ...
     </cre:P_SDR_LINES_TBL_ITEM>
   </cre:P_SDR_LINES_TBL>
   <cre:P_SDR_CUST_TBL>
     ...
   </cre:P_SDR_CUST_TBL>
  </cre:InputParameters>>
</soapenv:Body>
</soapenv:Envelope>
```

A Sample SOAP Response

The following example shows a SOAP response for a PL/SQL service:

```
<env:Envelope xmlns:env=""http://schemas.xmlsoap.org/soap/envelope/">
 <env:Header/>
  <env:Body>
    <OutputParameters xmlns="http://xmlns.oracle.
com/apps/ozf/soaprovider/plsql/ozf_sd_request_pub/create_sd_request/">
     <X_RETURN_STATUS>S</X_RETURN_STATUS>
     <X_MSG_COUNT>23</X_MSG_COUNT>
     <X_MSG_DATA xsi:nil="true" xmlns:xsi="http://www.w3.
org/2001/XMLSchema-instance"/>
     <X_REQUEST_HEADER_ID>162</X_REQUEST_HEADER_ID>
   </OutputParameters>
  </env:Body>
</env:Envelope>
```

A Sample Fault SOAP Response

The SOAP `Fault` element is used to carry error and status information within a SOAP message.

For example, the following fault response message indicates that the service is not deployed:

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
 <env:Header/>
  <env:Body>
   <Fault xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
      <faultcode xmlns="">SOAP-ENV:Server</faultcode>
      <faultstring xmlns="">Service is not deployed.</faultstring>
   </env:Fault>
 </env:Body>
</env:Envelope>
```

Examples of SOAP Messages Through Web Service Provider

To better understand SOAP request and response messages for business service object exposed to Web services through Web Service Provider, the following sample SOAP messages are described in this section:

- A Sample SOAP Request for Business Service Object, page 2-39
- A Sample SOAP Response for Business Service Object, page 2-40
- A Sample Fault SOAP Response for Business Service Object, page 2-41

A Sample SOAP Request for Business Service Object

The following example shows a valid SOAP request for business service object:

```
<soapenv:Envelope xmlns:ser="http://xmlns.oracle.
com/apps/fnd/ServiceBean"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ws="
http://xmlns.oracle.com/apps/fnd/rep/ws">
  <soapenv:Header>
   <wsse:Security soapenv:mustUnderstand="1" xmlns:wsse="http://docs.
oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
   <wsse:UsernameToken wsu:Id="UsernameToken-22948433" xmlns:wsu="http:
//docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd">
      <wsse:Username>sysadmin</wsse:Username>
      <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-username-token-profile-1.0#PasswordText">password</wsse:
Password>
    </wsse:UsernameToken>
   </wsse:Security>
  <ser:ServiceBean_Header>
    <ser:RESPONSIBILITY_NAME>System Administrator</ser:
RESPONSIBILITY_NAME>
    <ser:RESPONSIBILITY_APPL_NAME>sysadmin</ser:
RESPONSIBILITY_APPL_NAME>
    <ser:SECURITY_GROUP_NAME>standard</ser:SECURITY_GROUP_NAME>
    <ser:NLS_LANGUAGE>american</ser:NLS_LANGUAGE>
    <ser:ORG_ID>202</ser:ORG_ID>
  </ser:ServiceBean_Header>
 </soapenv:Header>
 <soapenv:Body>
   <ws:IntegrationRepositoryService_GetInterfaceByType>
     <interfaceType>XMLGATEWAY</interfaceType>
   </ws:IntegrationRepositoryService_GetInterfaceByType>
  </soapenv:Body>
</soapenv:Envelope>
```

A Sample SOAP Response for Business Service Object

The following example shows a valid SOAP response for business service object:

```
<soapenv:Envelope xmlns:env=http://schemas.xmlsoap.org/soap/envelope/">
  <env:Header/>
  <env:Body>
  <oans:IntegrationRepositoryService_GetInterfaceByType_Response
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="
http://www.w3.org/2001/XMLSchema"
   xmlns:oans="http://xmlns.oracle.com/apps/fnd/rep/ws">
      <interfaceClass>
        <ClassId>906</ClassId>
   <ClassName>XMLGATEWAY:CLN:SHIP_ORDER_CONFIRM_OUT</ClassName>
     <IrepName>CLN:SHIP_ORDER_CONFIRM_OUT</IrepName>
   <SecurityGroupId xsi:nil="true"/>
   <ClassType>XMLGATEWAY</ClassType>
         <ProductCode>cln</ProductCode>
   <ImplementationName xsi:nil="true"/>
   <DeployedFlag>N</DeployedFlag>
   <GeneratedFlag>N</GeneratedFlag>
   <CompatibilityFlag>N</CompatibilityFlag>
     <AssocClassId xsi:nil="true"/>
   <ScopeType>PUBLIC</ScopeType>
   <LifecycleMode>ACTIVE</LifecycleMode>
   <SourceFileProduct>CLN</SourceFileProduct>
         ...

      <InterfaceFunction>
       ...
      </InterfaceFunction>
    </InterfaceClass>
  </oans:IntegrationRepositoryService_GetInterfaceByType_Response>
 </env:Body>
</env:Envelope>
```

A Sample Fault SOAP Response for Business Service Object

The SOAP `Fault` element is used to carry error and status information within a SOAP message.

For example, if a SOAP request message contains invalid header information or the header is missing from the request, then `Fault` element appears as a body entry in the response message as shown below for business service object:

```
<env:Envelope xmlns:env=http://schemas.xmlsoap.org/soap/envelope/">
 <env:Header/>
  <env:Body>
   <Fault xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
      <faultcode>SOAP-ENV:Client</faultcode>
      <faultstring>InvalidHeader: Invalid or missing header in request.
</faultstring>
   </env:Fault>
 </env:Body>
</env:Envelope>
```

Understanding REST Messages

Based on REST architecture, the REST message uses HTTP header and method POST to create or update Oracle E-Business Suite data through a service provider.

Supporting XML and JSON Message Formats

Unlike SOAP message completely based on XML format, REST messages can process both XML and non-XML formats such as JSON.

Note: Only Jackson JSON format is supported in this release. Other JSON formats, like Google GSON are not supported.

- **XML**, like HTML, organizes information by nesting angle-bracketed tag pairs (< or >).
- Compared to XML, **JSON** is light weight with faster parsing results. It is a simple text-based message format that is often used with REST services.

  It uses curly brackets ({ or }) to hierarchically structure information.

REST Message Structure

A REST request is a simple HTTP request which includes the following elements:

- Header

  This element defines the operating parameters of an HTTP transaction.

  REST service user credentials can be passed in HTTP header. For more information on REST service security, see REST Security Header, page 2-43.

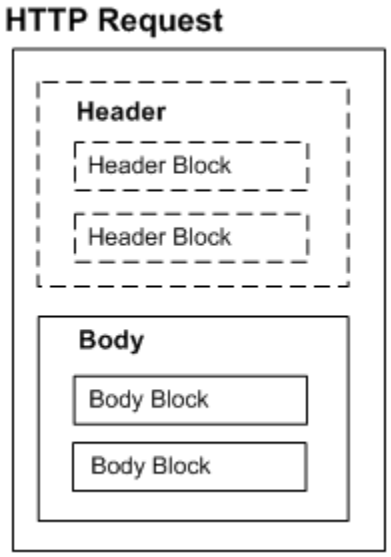
- Body

  This element defines the main messages or resources.

  'RESTHeader' element can be included in HTTP body to set applications context values if they are required in invoking the REST service.

  For more information on setting applications context, see REST Header for Applications Context, page 2-43.

The following diagram depicts the structure of a REST message:



REST Security Header

User credentials must be authenticated based on either one of the following methods:

- **HTTP Basic Authentication**

  In this security model, ***username and password*** should be provided as input data in HTTP header as part of the REST request message. When the REST service receives the request, the user credentials (username and password) will be routed to LoginModule for authentication and authorization. The LoginModule in turn extracts the credentials from HTTP header, authenticates user against Oracle E-Business Suite user table, and establishes the identity for the authenticated user.

  If user credentials are validated and applications context required for the REST service to be invoked can be initialized, the REST service can be invoked.

  For more information about HTTP Basic Authentication security, see HTTP Basic Authentication, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

- **Token Based Authentication**

  Instead of passing an associated password for the user, a ***security token*** can be passed as user credentials in place of password.

  When a user tries to log on to a server with multiple requests, instead of authenticating the user each time with username and password, a unique access token (such as Oracle E-Business Suite session ID) may be sent along with username in HTTP header. Oracle E-Business Suite session ID can be obtained by making call to Login service. The LoginModule will interpret and extract the token from the HTTP header, and validate the subject or username with token in the subsequent requests for authentication.

  If user credentials are validated and applications context required for the REST service to be invoked can be initialized, the service can be invoked.

  For more information on setting applications context, see REST Header for Applications Context, page 2-43.

  For more information about token based security, see Token Based Authentication, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

REST Header for Applications Context

Some Oracle E-Business Suite APIs require applications context values to be passed before they can be invoked. These context values including Responsibility, RespApplication. SecurityGroup, NLSLanguage, and Org_Id may be included in the `RESTHeader` element as part of the HTTP body.

Optional Context Values in Token Based Security

Context header values are optional. If the context values are not passed while using

token based security, the previously passed values will be used. If context values are passed, newly passed values will override the ones set previously for the given token.

The following REST message in XML format shows the `RESTHeader` element printed in bold:

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<TESTUSERNAME_Input xmlns="http://xmlns.oracle.
com/apps/fnd/rest/FndUserSvc/testusername/">
    <RESTHeader xmlns="http://xmlns.oracle.
com/apps/fnd/rest/FndUserSvc/header">
<Responsibility>SYSTEM_ADMINISTRATOR</Responsibility>
     <RespApplication></RespApplication>
     <SecurityGroup></SecurityGroup>
     <NLSLanguage>AMERICAN</NLSLanguage>
     <Org_Id>/Org_Id>
    </RESTHeader>
    <InputParameters>
   <X_USER_NAME>sysadmin</X_USER_NAME>
    </InputParameters>
</TESTUSERNAME_Input>
```

Constructing Payload from WADL Description

Based on the resources information in a WADL description, you can compile an input payload before invoking a REST service.

Use the following steps to compile an input payload:

1. In the Integration Repository, search and locate the deployed REST service that you want to use.

2. Click the **View WADL** link in the REST Web Service tab. The following WADL description appears:

```
<xml version="1.0" encoding="UTF-8" standalone="no" ?>
<application xmlns:tns="http://xmlns.oracle.
com/apps/fnd/soaprovider/plsql/rest/fnd_user_pkg/" xmlns="http:
//wadl.dev.java.net/2009/02" xmlns:tns1="http://xmlns.oracle.
com/apps/fnd/rest/FndUserSvc/testusername/" name="FND_USER_PKG"
targetNamespace="http://xmlns.oracle.
com/apps/fnd/soaprovider/plsql/rest/fnd_user_pkg/">
<grammars>
    <include xmlns="http://www.w3.org/2001/XMLSchema" href="https:
//<hostname>:<port>/webservices/rest/FndUserSvc/?
XSD=TESTUSERNAME_SYNCH_TYPEDEF.xsd" />
</grammars><resources base="http://<hostname>:
<port>/webservices/rest/FndUserSvc/"><resource path="/testusername/"
>
   <method id="GET" name="POST">
    <request>
     <representation mediaType="application/xml" type="tns1:
InputParameters" />
     <representation mediaType="application/json" type="tns1:
InputParameters" />
    </request>
    <response>
     <representation mediaType="application/xml" type="tns1:
OutputParameters" />
     <representation mediaType="application/json" type="tns1:
OutputParameters" />
    </response>
   </method>
  </resource>
 </resources>
</application>
```

3. Locate the schema information (.XSD) for the Test User Name (TESTUSERNAME) service operation from the WADL description. The XSD for the operation TESTUSERNAME in the WADL would be:

   ```
   http://<hostname>:<port>/webservices/rest/FndUserSvc/?
   XSD=TESTUSERNAME_SYNCH_TYPEDEF.xsd
   ```

   > **Note:** The schema information for the service operation can also be constructed by concatenating the values of the following elements from the WADL description:
   >
   > - `<resources base="http://<hostname>:<port>/webservices/rest/FndUserSvc/">`
   > - `<resource path="/testusername/">`

4. Construct the payload of the service by using any XSD to XML conversion tools to get the payload information.

Once the payload is compiled, it can be used to invoke the TESTUSERNAME REST service operation. The request, response, and fault messages with both XML and JSON formats are listed in the following table:

Input Payload (Request Message)

XML-based REST Message

```
<?xml version="1.0" encoding="UTF-8" ?>
<TESTUSERNAME_Input xmlns="http://xmlns.oracle.
com/apps/fnd/rest/FndUserSvc/testusername/">
 <RESTHeader xmlns="http://xmlns.oracle.
com/apps/fnd/rest/FndUserSvc/header">
     <Responsibility>SYSTEM_ADMINISTRATOR</Responsibility>
     <RespApplication></RespApplication>
     <SecurityGroupE></SecurityGroup>
     <NLSLanguage></NLSLanguage>
     <Org_Id></Org_Id>
   </RESTHeader>
  <InputParameters>
  <X_USER_NAME>sysadmin</X_USER_NAME>
 </InputParameters>
</TESTUSERNAME_Input>
```

JSON-based REST Message

```
{"TESTUSERNAME_Input":{
"@xmlns":"http://xmlns.oracle.
com/apps/fnd/rest/FndUserSvc/testusername/",
"RESTHeader":{
"@xmlns":"http://xmlns.oracle.
com/apps/fnd/rest/FndUserSvc/header",
"Responsibility":"SYSTEM_ADMINISTRATOR",
"RespApplication":"SYSADMIN",
"SecurityGroup":"STANDARD",
"NLSLanguage":"AMERICAN",
"Org_Id":"202"
},
"InputParameters":{
"X_USER_NAME":"operations"
}
}}
```

Response

XML-based REST Message

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<OutputParameters xmlns:xsi="http://www.w3.
org/2001/XMLSchema-instance"
 xmlns="http://xmlns.oracle.
com/apps/fnd/rest/FndUserSvc/testusername/">
<X_USER_NAME>2</X_USER_NAME>
</OutputParameters>
```

JSON-based REST Message

```
{
  "OutputParameters" : {
    "@xmlns:xsi" : "http://www.w3.org/2001/XMLSchema-
instance",
    "@xmlns" : "http://xmlns.oracle.
com/apps/fnd/rest/fndGlobalSvc/user_id/",
    "TESTUSERNAME" : "2"
  }
}
```

REST Messages with XML and JSON Formats	

Error Response	XML-based REST Message
	<ISGServiceFault>
	<Code>IRepAccessError</Code>
	<Message>This is a sample Fault Message. Message will vary depending on fault condition</Message>
	<Resolution>Check the server logs for details</Resolution>
	<ServiceDetails>
	<ServiceName>FndUserSvc</ServiceName>
	<OperationName>testusername</OperationName>
	<InstanceId>0</InstanceId>
	</ServiceDetails>
	</ISGServiceFault>
	
	JSON-based REST Message
	{
	"ISGServiceFault": {
	"Code": "IRepAccessError",
	"Message": "Sample Fault Message. Will vary depending on fault condition",
	"Resolution": "Check the server logs for details",
	"ServiceDetails": {
	"ServiceName": "FndUserSvc",
	"OperationName": "testusername",
	"InstanceId": "0"
	}
	}
	}

For more examples of REST messages used in OZF_SD_REQUEST_PUB service invocation, see Examples of REST Messages, page 2-47.

Examples of REST Messages

To better understand REST request and response messages received through Oracle E-Business Suite, the following sample REST messages are described in this section:

- A Sample XML-based REST Request, page 2-47
- A Sample JSON-based REST Request, page 2-52
- A Sample XML-based REST Response, page 2-54
- A Sample JSON-based REST Response, page 2-56
- Samples of XML-based Fault Responses, page 2-56

A Sample XML-based REST Request

This section includes sample REST requests to create a ship and debit request using a

PL/SQL service, and requests to update, insert, and delete data in an open interface table.

- **A synchronous request for a PL/SQL service**

  The following example shows a synchronous XML-based REST request for a PL/SQL service (OZF_SD_REQUEST_PUB API):

```
<?xml version="1.0" encoding="UTF-8" ?>
<CREATE_SD_REQUEST_Input xmlns:xsi="http://www.w3.
org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://xmlns.oracle.
com/apps/ozf/rest/ozfsdrequestpubsvc/create_sd_request/xsd/OZF_SD_RE
QUEST_PUB_CREATEREQUEST.xsd"
      xmlns="http://xmlns.oracle.
com/apps/ozf/rest/ozfsdrequestpubsvc/create_sd_request/">
 <RESTHeader xmlns="http://xmlns.oracle.
com/apps/fnd/rest/ozfsdrequestpubsvc/header">
     <Responsibility>SYSTEM_ADMINISTRATOR</Responsibility>
     <RespApplication></RespApplication>
     <SecurityGroupE></SecurityGroup>
     <NLSLanguage></NLSLanguage>
     <Org_Id></Org_Id>
   </RESTHeader>
  <InputParameters>
  <P_API_VERSION_NUMBER>1.0</P_API_VERSION_NUMBER>
  <P_INIT_MSG_LIST>T</P_INIT_MSG_LIST>
     <P_COMMIT>F</P_COMMIT>
  <P_VALIDATION_LEVEL>100</P_VALIDATION_LEVEL>
      <P_SDR_HDR_REC>
        <REQUEST_NUMBER>SDR-CREATE-BPEL1</REQUEST_NUMBER>
        <REQUEST_START_DATE>2008-08-18T12:00:00</REQUEST_START_DATE>
      <REQUEST_END_DATE>2008-10-18T12:00:00</REQUEST_END_DATE>>
        <USER_STATUS_ID>1701</USER_STATUS_ID>
        <REQUEST_OUTCOME>IN_PROGRESS</REQUEST_OUTCOME>
        <REQUEST_CURRENCY_CODE>USD</EQUEST_CURRENCY_CODE>
          <SUPPLIER_ID>601</SUPPLIER_ID>
          <SUPPLIER_SITE_ID>1415</SUPPLIER_SITE_ID>
         <REQUESTOR_ID>xxxxxxxxx</REQUESTOR_ID>
      <ASSIGNEE_RESOURCE_ID>xxxxxxxxx</ASSIGNEE_RESOURCE_ID>
      <ORG_ID>204</ORG_ID>
      <ACCRUAL_TYPE>SUPPLIER</ACCRUAL_TYPE>
      <REQUEST_DESCRIPTION>Create</REQUEST_DESCRIPTION>
    <SUPPLIER_CONTACT_EMAIL_ADDRESS>sdr.supplier@example.
com</SUPPLIER_CONTACT_EMAIL_ADDRESS>

<SUPPLIER_CONTACT_PHONE_NUMBER>2255</SUPPLIER_CONTACT_PHONE_NUMBER>
       <REQUEST_TYPE_SETUP_ID>400</REQUEST_TYPE_SETUP_ID>
         <REQUEST_BASIS>Y</REQUEST_BASIS>
         <USER_ID>xxxxxxx</USER_ID>
   </P_SDR_HDR_REC>
   <P_SDR_LINES_TBL>
     <P_SDR_LINES_TBL_ITEM>
        <PRODUCT_CONTEXT>PRODUCT</PRODUCT_CONTEXT>
    <INVENTORY_ITEM_ID>2155</INVENTORY_ITEM_ID>
    <ITEM_UOM>Ea</ITEM_UOM>
    <REQUESTED_DISCOUNT_TYPE>%</REQUESTED_DISCOUNT_TYPE>
    <REQUESTED_DISCOUNT_VALUE>20</REQUESTED_DISCOUNT_VALUE>
    <COST_BASIS>200</COST_BASIS>
    <MAX_QTY>200</MAX_QTY>
    <DESIGN_WIN>200</DESIGN_WIN>
    <APPROVED_DISCOUNT_TYPE>%</APPROVED_DISCOUNT_TYPE>
    <APPROVED_DISCOUNT_VALUE>20</APPROVED_DISCOUNT_VALUE>
    <APPROVED_MAX_QTY>200</APPROVED_MAX_QTY>
    <VENDOR_APPROVED_FLAG>Y</VENDOR_APPROVED_FLAG>
    <PRODUCT_COST_CURRENCY>USD</PRODUCT_COST_CURRENCY>
    <END_CUSTOMER_CURRENCY>USD</END_CUSTOMER_CURRENCY>
     </P_SDR_LINES_TBL_ITEM>
   </P_SDR_LINES_TBL>
   <P_SDR_CUST_TBL>
  <P_SDR_CUST_TBL_ITEM>
   <CUST_ACCOUNT_ID>1290</CUST_ACCOUNT_ID>
   <PARTY_ID>1290</PARTY_ID>
```

```
<SITE_USE_ID>10479</SITE_USE_ID>
   <CUST_USAGE_CODE>BILL_TO</CUST_USAGE_CODE>
   <END_CUSTOMER_FLAG>N</END_CUSTOMER_FLAG>
  </P_SDR_CUST_TBL_ITEM>
  <P_SDR_CUST_TBL_ITEM>
   <CUST_ACCOUNT_ID>1287</CUST_ACCOUNT_ID>
   <PARTY_ID>1287</PARTY_ID>
   <SITE_USE_ID>1418</SITE_USE_ID>
   <CUST_USAGE_CODE>CUSTOMER</CUST_USAGE_CODE>
   <END_CUSTOMER_FLAG>Y</END_CUSTOMER_FLAG>
  </P_SDR_CUST_TBL_ITEM>
   </P_SDR_CUST_TBL>
  </InputParameters>
</CREATE_SD_REQUEST_Input>
```

- **A request for updating records in an Open Interface Table**

  The following example shows a request message to **update** (PUT HTTP method) the open interface table RA_INTERFACE_LINES_ALL contained in the "AR Autoinvoice" Open Interface (RAXMTR) with Inbound direction:

  In this example, the HTTP request consists of more than one Update requests. The response message for each Update request should include status, message, and number of records updated. This request also includes `Select`; therefore, the response message returns values for the provided columns, INTERFACE_LINE_ID and BATCH_SOURCE_NAME in the `Select` statement.

```
 PUT http://host:
port/webservices/rest/autoinvoice/RA_INTERFACE_LINES_ALL
 Content-Type: application/xml

 <?xml version="1.0" encoding="UTF-8" ?>
<RA_INTERFACE_LINES_ALL_Input xmlns="http://xmlns.oracle.
com/apps/ar/rest/autoinvoice/RA_INTERFACE_LINES_ALL">
  <RESTHeader xmlns="http://xmlns.oracle.com/apps/fnd/rest/header">
   <Responsibility>RECEIVABLES_VISION_OPERATIONS</Responsibility>
     <RespApplication>AR</RespApplication>
     <SecurityGroupE>STANDARD</SecurityGroup>
     <NLSLanguage>AMERICAN</NLSLanguage>
     <Org_Id>204</Org_Id>
   </RESTHeader><Select>INTERFACE_LINE_ID,
BATCH_SOURCE_NAME</Select>
  <InputParameters>
   <Update>
   <Filter>BATCH_SOURCE_NAME==ICS-01;ERROR_FLAG==T;
QUANTITY=le=100</Filter>
   <RA_INTERFACE_LINES_ALL_REC>
    <REQUEST_ID></REQUEST_ID>
    <ERROR_FLAG />
    <PROCESS_FLAG>0</PROCESS_FLAG>
   </Update>
   <Update>
   <Filter>BATCH_SOURCE_NAME==ICS-01;ERROR_FLAG==T;
QUANTITY=gt=100</Filter>
   <RA_INTERFACE_LINES_ALL_REC>
    <PROCESS_FLAG>4</PROCESS_FLAG>
   </RA_INTERFACE_LINES_ALL_REC>
   </Update>
  </InputParameters>
</RA_INTERFACE_LINES_ALL_Input>
```

- **A request for inserting records in an Open Interface Table**

This example explains the sample request to **insert** (POST HTTP method) data in the same open interface table RA_INTERFACE_LINES_ALL contained in the same "AR Autoinvoice" (RAXMTR) with Inbound direction.

This request consists of two records to be inserted in the selected column names, INTERFACE_LINE_ID and BATCH_SOURCE_NAME, as shown below in the Select parameter. Similar to the update sample request mentioned previously, the response message returns values for the same selected columns.

```
 PUT http://host:
port/webservices/rest/autoinvoice/RA_INTERFACE_LINES_ALL
 Content-Type: application/xml

 <?xml version="1.0" encoding="UTF-8" ?>
<RA_INTERFACE_LINES_ALL_Input xmlns="http://xmlns.oracle.
com/apps/ar/rest/autoinvoice/RA_INTERFACE_LINES_ALL">
  <RESTHeader xmlns="http://xmlns.oracle.com/apps/fnd/rest/header">
   <Responsibility>RECEIVABLES_VISION_OPERATIONS</Responsibility>
     <RespApplication>AR</RespApplication>
     <SecurityGroupE>STANDARD</SecurityGroup>
     <NLSLanguage>AMERICAN</NLSLanguage>
     <Org_Id>204</Org_Id>
   </RESTHeader><Select>INTERFACE_LINE_ID,
BATCH_SOURCE_NAME</Select>
  <InputParameters>
   <RA_INTERFACE_LINES_ALL_REC>
    <REQUEST_ID></REQUEST_ID>
    <BATCH_SOURCE_NAME>ICS-1</BATCH_SOURCE_NAME>
    <RELATED_TRX_NUMBER>101</RELATED_TRX_NUMBER>
          <INTERFACE_LINE_ID>11</INTERFACE_LINE_ID>
           ...
   </RA_INTERFACE_LINES_ALL_REC>
   <RA_INTERFACE_LINES_ALL_REC>
          <REQUEST_ID></REQUEST_ID>
    <BATCH_SOURCE_NAME>ICS-1</BATCH_SOURCE_NAME>
    <RELATED_TRX_NUMBER>102</RELATED_TRX_NUMBER>
          <INTERFACE_LINE_ID>12</INTERFACE_LINE_ID>
          ...
      </RA_INTERFACE_LINES_ALL_REC>
 </InputParameters>
</RA_INTERFACE_LINES_ALL_Input>
```

- **A request for deleting records in an Open Interface Table**

  The same open interface table RA_INTERFACE_LINES_ALL contained in "AR Autoinvoice" (RAXMTR) is used in this sample request for a Delete operation (DELETE HTTP method). This operation should accept filter criteria as query parameter.

  For example, the following request should delete records in RA_INTERFACE_LINES_ALL if the records satisfy the condition ORG_ID = 204, TRX_DATE < 27/04/2016, and PROCESS_FLAG = 4.

  Note that the query is based on Feed Item Query Language (FIQL) or Resource Query Language (RQL).

```
http://host:
port/webservices/rest/autoinvoice/RA_INTERFACE_LINES_ALL? filter
=ORG_ID==204;TRX_DATE=lt=2016-04-27T00:00:00.000%2B00:00;
PROCESS_FLAG==4
```

The response message includes Status, Message and Number of records deleted (`DeleteCount`).

A Sample JSON-based REST Request

The following example shows a synchronous JSON-based POST request for the same PL/SQL service (OZF_SD_REQUEST_PUB API):

> **Note:** Only Jackson JSON format is supported in this release. Other JSON formats, like Google GSON are not supported.

```
{
  "CREATE_SD_REQUEST_Input": {
    "@xmlns": "http://xmlns.oracle.
com/apps/ozf/rest/ozfsdrequestpubsvc/create_sd_request/",
    "RESTHeader": {
      "@xmlns": "http://xmlns.oracle.
com/apps/fnd/rest/ozfsdrequestpubsvc/header",
      "Responsibility": "SYSTEM_ADMINISTRATOR"
    },
    "InputParameters": {
      "P_API_VERSION_NUMBER": "1.0",
      "P_INIT_MSG_LIST": "T",
      "P_COMMIT": "F",
      "P_VALIDATION_LEVEL": "100",
      "P_SDR_HDR_REC": {
        "REQUEST_NUMBER": "SDR-CREATE-BPEL1",
        "REQUEST_START_DATE": "2008-08-18T12:00:00",
        "REQUEST_END_DATE": "2008-10-18T12:00:00",
        "USER_STATUS_ID": "1701",
        "REQUEST_OUTCOME": "IN_PROGRESS",
        "REQUEST_CURRENCY_CODE": "USD",
        "SUPPLIER_ID": "601",
        "SUPPLIER_SITE_ID": "1415",
        "REQUESTOR_ID": "xxxxxxxxx",
        "ASSIGNEE_RESOURCE_ID": "xxxxxxxxx",
        "ORG_ID": "204",
        "ACCRUAL_TYPE": "SUPPLIER",
        "REQUEST_DESCRIPTION": "Create",
        "SUPPLIER_CONTACT_EMAIL_ADDRESS": "sdr.supplier@example.com",
        "SUPPLIER_CONTACT_PHONE_NUMBER": "2255",
        "REQUEST_TYPE_SETUP_ID": "400",
        "REQUEST_BASIS": "Y",
        "USER_ID": "xxxxxxxx"
      },
      "P_SDR_LINES_TBL": {
        "P_SDR_LINES_TBL_ITEM": {
          "PRODUCT_CONTEXT": "PRODUCT",
          "INVENTORY_ITEM_ID": "2155",
          "ITEM_UOM": "Ea",
          "REQUESTED_DISCOUNT_TYPE": "%",
          "REQUESTED_DISCOUNT_VALUE": "20",
          "COST_BASIS": "200",
          "MAX_QTY": "200",
          "DESIGN_WIN": "200",
          "APPROVED_DISCOUNT_TYPE": "%",
          "APPROVED_DISCOUNT_VALUE": "20",
          "APPROVED_MAX_QTY": "200",
          "VENDOR_APPROVED_FLAG": "Y",
          "PRODUCT_COST_CURRENCY": "USD",
          "END_CUSTOMER_CURRENCY": "USD"
        }
      },
      "P_SDR_CUST_TBL": {
        "P_SDR_CUST_TBL_ITEM": [
          {
            "CUST_ACCOUNT_ID": "1290",
            "PARTY_ID": "1290",
            "SITE_USE_ID": "10479",
            "CUST_USAGE_CODE": "BILL_TO",
            "END_CUSTOMER_FLAG": "N"
          },
          {
            "CUST_ACCOUNT_ID": "1287",
            "PARTY_ID": "1287",
            "SITE_USE_ID": "1418",
            "CUST_USAGE_CODE": "CUSTOMER",
```

```
"END_CUSTOMER_FLAG": "Y"
          }
        ]
      }
    }
  }
}
```

The following example shows a JSON-based GET request for a Java Bean Service called REST Service Locator:

```
Request Headers
           Authorization: Basic xxxxxxxxxxxxxxxxxxxxxxx
           Accept: application/json
           Content-Language: en-US
```

> **Note:** For GET requests, JSON is the default output response format. Use `Accept` header `application/xml` to receive response in XML format. If `Content-Type` header is sent in GET HTTP request, it will be ignored.

A Sample XML-based REST Response

The following example shows an XML-based REST response for the OZF_SD_REQUEST_PUB API service:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<OutputParameters
xmlns:xsl=""http://www.w3.org/2001/XMLSchema-instance" xmlns="http:
//xmlns.oracle.com/apps/ozf/rest/ozfsdrequestpubsvc/create_sd_request/">
     <X_RETURN_STATUS>E</X_RETURN_STATUS>
     <X_MSG_COUNT>1</X_MSG_COUNT>>
   <X_MSG_DATA>The Organization Id provided is invalid, please provide a
valid Organization Id.</X_MSG_DATA>
     <X_REQUEST_HEADER_IDxsi:nil="true"/>
</OutputParameters>
```

Additionally, the following example shows a response message for the HTTP request mentioned earlier for the Open Interface Table "AR Autoinvoice". In response to the **Update** request for the INTERFACE_LINE_ID and BATCH_SOURCE_NAME columns in the RA_INTERFACE_LINES_ALL table, the response message includes status, message, and number of records updated (`<UpdateCount>`) for each Update request identified by Index in the order of occurrence in the request. When the update is successful, this response message also returns values for INTERFACE_LINE_ID and BATCH_SOURCE_NAME mentioned in the `Select` statement in the request.

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<OutputParameters xmlns="http://xmlns.oracle.
com/apps/ar/rest/autoinvoice/RA_INTERFACE_LINES_ALL">
 <Summary>
 <SuccessCount>5</SuccessCount>
 <ErrorCount>1</ErrorCount>
 </Summary>
 <Result><Index>1</Index> <Status>SUCCESS</Status>
 <Message></Message>
 <UpdateCount>2</UpdateCount>
 <RA_INTERFACE_LINES_ALL_REC>
  <INTERFACE_LINE_ID>11</INTERFACE_LINE_ID>
  <BATCH_SOURCE_NAME>ICS-1</BATCH_SOURCE_NAME>
   </RA_INTERFACE_LINES_ALL_REC>
   <RA_INTERFACE_LINES_ALL_REC>
      <INTERFACE_LINE_ID>12</INTERFACE_LINE_ID>
  <BATCH_SOURCE_NAME>ICS-1</BATCH_SOURCE_NAME>
   </RA_INTERFACE_LINES_ALL_REC>
 </Output>
 <Output><Index>2</Index><Status>ERROR</Status>
 <Message>Invalid date .. </Message>
 <UpdateCount>0</UpdateCount>
   <RA_INTERFACE_LINES_ALL_REC>
 </Output>
 </Result>
</OutputParameters>
```

Similarly, for the request of **inserting** two records in the same open interface table RA_INTERFACE_LINES_ALL as described earlier, the response message returns values for the same selected columns INTERFACE_LINE_ID and BATCH_SOURCE_NAME as shown in the following:

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<OutputParameters xmlns="http://xmlns.oracle.
com/apps/ar/rest/autoinvoice/RA_INTERFACE_LINES_ALL">
 <Summary>
 <SuccessCount>15</SuccessCount>
 <ErrorCount>2</ErrorCount>
 </Summary>
 <Result>
  <Output><Index>1</Index>
    <Status>SUCCESS</Status>
   <Message></Message>
  <RA_INTERFACE_LINES_ALL_REC>
  <INTERFACE_LINE_ID>11</INTERFACE_LINE_ID>
  <BATCH_SOURCE_NAME>ICS-1</BATCH_SOURCE_NAME>
    </RA_INTERFACE_LINES_ALL_REC>
   </Output>
   <Output><Index>2</Index>
    <Status>ERROR</Status>
    <Message>Invalid date .. </Message>
   <RA_INTERFACE_LINES_ALL_REC>
      <INTERFACE_LINE_ID>12</INTERFACE_LINE_ID>
  <BATCH_SOURCE_NAME>ICS-1</BATCH_SOURCE_NAME>
   </RA_INTERFACE_LINES_ALL_REC>
 </Output>
 </Result>
</OutputParameters>
```

In response to the **Delete** request explained earlier, the response message includes Status, Message and Number of records being deleted (`DeleteCount`):

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<OutputParameters xmlns="http://xmlns.oracle.
com/apps/ar/rest/autoinvoice/RA_INTERFACE_LINES_ALL">
 <Result>
    <Status>SUCCESS</Status>
  <Message></Message>
    <DeleteCount>2</DeleteCount>
 </Result>
</OutputParameters>
```

A Sample JSON-based REST Response

The following example shows a JSON-based REST response for the OZF_SD_REQUEST_PUB API service with POST method:

```
{
  "OutputParameters" : {
    "@xmlns:xsi" : "http://www.w3.org/2001/XMLSchema-instance",
    "@xmlns" : "http://xmlns.oracle.
com/apps/ozf/rest/ozfsdrequestpubsvc/create_sd_request/",
    "X_RETURN_STATUS" : "E",
    "X_MSG_COUNT" : "1",
    "X_MSG_DATA" : "The Organization Id provided is invalid, please
provide a valid Organization Id.",
    "X_REQUEST_HEADER_ID" : {
      "@xsi:nil" : "true"
    }
  }
```

The following example shows a JSON-based REST response for the REST Service Locator (getRestInterface service operation) service with GET method:

```
   {
                  "OutputParameters" : {
                    "EbsRestServiceBean" : [ {
                   "alternateAlias" : "plsql/PLSQL:FND_PROFILE",
                   "serviceAlias" : "NotAnything",
                   "serviceName" : "PLSQL:FND_PROFILE",
                   "wadlUrl" : "http://<hostname>:
<port>/webservices/rest/profile?WADL"
                    } ],
                "ControlBean" : [ {
                      "fields" : "",
                      "filter" : "",
                      "limit" : "",
                      "offset" : ""
                } ]
          }
        }
```

Samples of XML-based Fault Responses

The following sample shows the XML-based REST response message when XML is not well formed:

```
<ISGServiceFault>
 <Code>RequestParsingError</Code>
 <Message>SAXException in XmlRequestObject, while parsing XML request
The request could not be parsed correctly</Message>
 <Resolution>This may be due to malformed construction of the payload or
incorrectContent-Type header. Please check the wellformed-ness of
payload, matching Content-Type header of the http request and retry.
</Resolution>
 <ServiceDetails>
  <ServiceName>ozfsdrequestpubsvc</ServiceName>
  <OperationName>create_sd_request</OperationName>
  <InstanceId>0</InstanceId>
 </ServiceDetails>
</ISGServiceFault>
```

The following sample shows the XML-based REST response message when `RespApplication` (Responsibility Application short name) is invalid:

```
<ISGServiceFault>
 <Code>InvalidResponsibilityApplicationShortCode</Code>
 <Message>Responsibility short code is invalid System error while
processing the request</Message>
 <Resolution>Check the server logs for details</Resolution>
 <ServiceDetails>
  <ServiceName>ozfsdrequestpubsvc</ServiceName>
  <OperationName>get_text_number</OperationName>
  <InstanceId>0</InstanceId>
 </ServiceDetails>
</ISGServiceFault>
```

3

Using PL/SQL APIs as Web Services

Overview

Oracle E-Business Suite Integrated SOA Gateway allows you to use PL/SQL application programming interfaces (APIs) to insert or update data in Oracle E-Business Suite. APIs are stored procedures that let you update or retrieve data from Oracle E-Business Suite.

Once a PL/SQL API interface definition is exposed as a Web service representing in WSDL URL, the generated Web service can be deployed from the Oracle Integration Repository to Oracle E-Business Suite application server. Services can then be exposed to customers through service provider and invoked through any of the Web service clients or orchestration tool including Oracle JDeveloper, Apache Axis, .NET Web Service Client, Oracle BPEL Process Manager, and Oracle Enterprise Service Bus (ESB).

For example, these deployed Web services can be orchestrated into a meaningful business process with service endpoints using a BPEL language. At run time, the BPEL process can be deployed to Oracle BPEL server or a third party BPEL server that can be consumed by customers.

In addition to SOAP services, PL/SQL APIs can be exposed as REST services. To better understand how to use each Web service in inserting or updating application data, detailed design-time and run-time tasks are discussed in this chapter. For the example described in the following sections, Oracle JDeveloper 10.1.3.3.0 is used as a design-time tool to create a SOA composite application with BPEL process and Oracle SOA Suite BPEL server 10.1.3.3.0 is used for the process deployment.

This chapter includes the following topics:

- Invoking a REST Service Using Token Based Authentication and JSON Payload, page 3-50

Using PL/SQL SOAP Services

BPEL Process Scenario

Take PL/SQL Supplier Ship and Debit Request API `OZF_SD_REQUEST_PUB` as an example to explain the BPEL process creation.

When the creation of a ship and debit request is received, the creation information including input ship and debit payload will be read and passed to create a ship and debit request. Once the request is created, the request number will then be returned to the requestor.

If the BPEL process is successfully executed after deployment, you should find a ship and debit request is created in the Oracle Order Management. The request number should be the same as the payload input value.

Prerequisites to Create a BPEL Process Using a PL/SQL Web Service

Before performing the design-time tasks for PL/SQL Web services, you need to ensure the following tasks are in place:

> **Note:** Before generating the Web service for a selected interface, you can also create a security grant to a specific user (such as "TRADEMGR") or user group if necessary to ensure the user has the access privilege to the interface.

- An integration repository administrator needs to successfully generate and deploy a Web service to the application server.
- An integration developer needs to locate and record the deployed WSDL URL for the PL/SQL exposed as a Web service.
- SOAHeader variables need to be populated for Web service authorization.

Please note that certain PL/SQL APIs exposed from Oracle E-Business Suite Integrated SOA Gateway take record types as input. Such APIs expect default values to be populated for parameters within these record types for successful execution.

The default values are `FND_API.G_MISS_CHAR` for characters, `FND_API.G_MISS_DATE` for dates, and `FND_API.G_MISS_NUM` for numbers. Oracle E-Business Suite Integrated SOA Gateway can default these values when the parameters within the record type are passed as nil values, for example, as shown below:

```
<PRICE_LIST_REC>
<ATTRIBUTE1 xsi:nil="true"/>
<ATTRIBUTE2 xsi:nil="true"/>
<ATTRIBUTE3 xsi:nil="true"/>
...
</PRICE_LIST_REC>
```

Deploying PL/SQL WSDL URL

An integration repository administrators must first create a Web service for a selected interface definition, and then deploy the service from Oracle Integration Repository to the application server.

For example, the administrator must perform the following steps before letting the integration developers use the deployed WSDL in creating a BPEL process:

1. To generate a Web service, locate the interface definition first (such as a PL/SQL interface `OZF_SD_REQUEST_PUB`) and click **Generate** in the SOAP Web Service tab of the interface details page.

   For detailed instruction on how to generate a Web service, see Generating SOAP Web Services, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

2. To deploy a generated Web service, select one authentication type and click **Deploy** in the SOAP Web Service tab of the interface details page to deploy the service.

   Once the service is successfully deployed, the selected authentication type will be displayed along with 'Deployed' Web Service Status. For more information on securing Web services with authentication types, see Managing Web Service Security, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

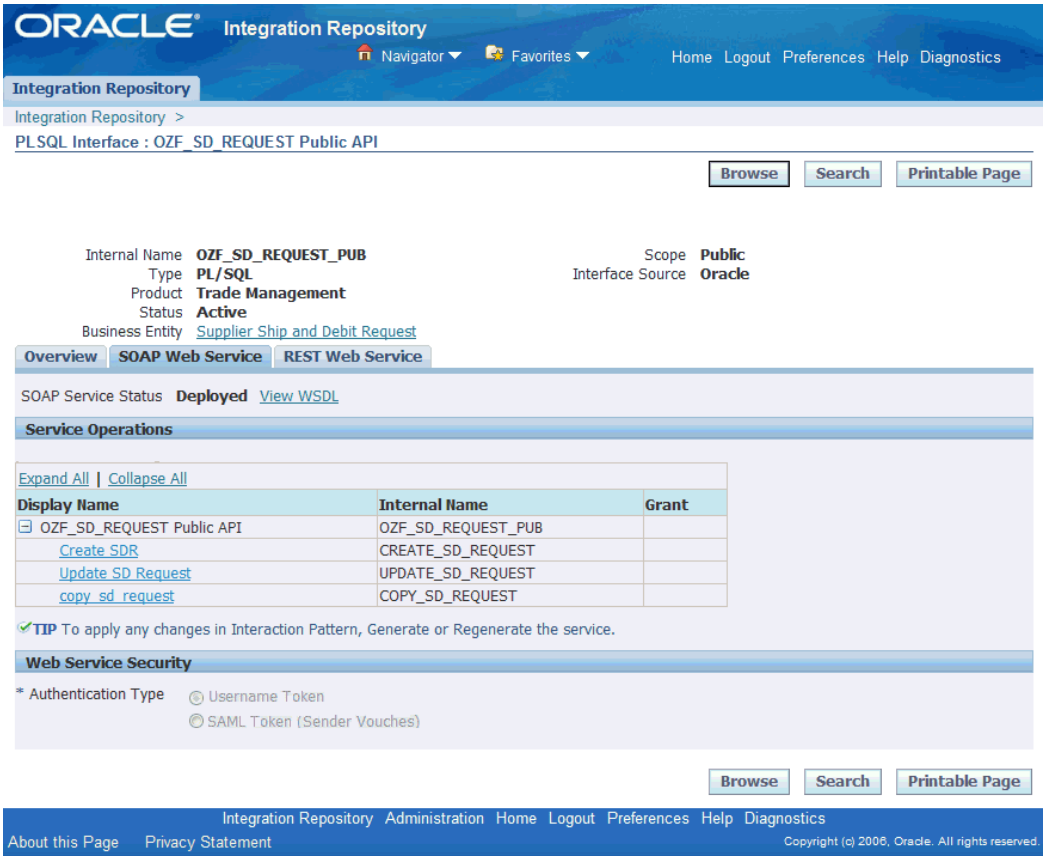
   For detailed instruction on how to deploy a Web service, see Deploying, Undeploying, and Redeploying SOAP Web Services, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

Searching and Recording WSDL URL

Apart from the required tasks performed by the administrators, an integration developer also needs to log on to the system to locate and record the deployed Web service WSDL URL for the interface that needs to be orchestrated into a meaningful business process in Oracle JDeveloper using BPEL language.

This can be done by clicking the **View WSDL** link in the interface details page to open a new window. Copy the WSDL URL from the new window. This URL will be used later in creating a partner link for the interface exposed as a Web service during the BPEL process creation at design time.

PL/SQL Interface Details Page: SOAP Web Service Tab with Deployed WSDL URL



How to search for an interface and review the interface details, see Searching and Viewing Integration Interfaces, page 2-2.

Setting Variables in SOAHeader for SOAP Request

You must populate certain variables in the BPEL process for SOAHeader elements to pass values that would be used to set application context during service execution. These SOAHeader elements for PL/SQL interface type are `Responsibility`, `RespApplication`, `SecurityGroup`, `NLSLanguage`, and `Org_Id`.

> **Note:** The user information is defined by the `wsseUsername` property passed within the security headers. Detailed instructions on how to pass the security headers along with the SOAP request, see Passing Values to Security Headers, page 3-10.

The expected values for these elements are described in the following table:

Header Variables and Expected Values for PL/SQL Interface Type

Element Name	Expected Value
`Responsibility`	`responsibility_key` (such as "SYSTEM_ADMINISTRATOR")
`RespApplication`	Application Short Name (such as "FND")
`SecurityGroup`	Security Group Key (such as "STANDARD")
`NLSLanguage`	NLS Language (such as "AMERICAN")
`Org_Id`	Org Id (such as "202")

Note: NLS Language and Org_Id are optional values to be passed.

- If the NLS Language element is specified, SOAP requests can be consumed in the language passed. All corresponding SOAP responses and error messages can also be returned in the same language. If no language is identified, then the default language of the user will be used.
- If a service execution is dependent on any particular organization, then you must pass the Org_Id element of that SOAP request.

The context information can be specified by configuring an Assign activity before the Invoke activity in the BPEL PM.

Detailed information on how to set SOAHeader for the SOAP request, see Assigning SOAHeader Parameters, page 3-25.

Using a PL/SQL Services at Design Time

Based on the single invoice creation scenario, the following design-time tasks are discussed in this chapter:

1. Create a new BPEL project, page 3-6

   Use this step to create a new BPEL project called `ShipDebitRequest.bpel` using an Synchronous BPEL Process template. This automatically creates two dummy activities - Receive and Reply - to receive input from a third party application and to reply output of the BPEL process back to the request application.

2. Create a Partner Link, page 3-8

   Use this step to create a ship and debit request in Oracle Order Management by using the Supplier Ship and Debit Request API `OZF_SD_REQUEST_PUB` exposed as Web service.

3. Add a Partner Link for File Adapter, page 3-12

   Use this step to synchronous read invoice header details passed from the first Assign activity.

4. Add Invoke activities, page 3-21

   Use this step to configure two Invoke activities in order to:

   - Point to the File Adapter to synchronous read invoice header details that is passed from the first Assign activity.
   - Point to the `OZF_SD_REQUEST_PUB` partner link to initiate the request creation with payload and transaction details received from the Assign activities.

5. Add Assign activities, page 3-24

   Use this step to configure Assign activities in order to pass request header details, payload information and request number to appropriate Invoke activities to facilitate the request creation. At the end, pass the request number to the request application through the dummy Reply activity.

For general information and basic concept of a BPEL process, see Understanding BPEL Business Processes, page D-1 and *Oracle BPEL Process Manager Developer's Guide* for details.

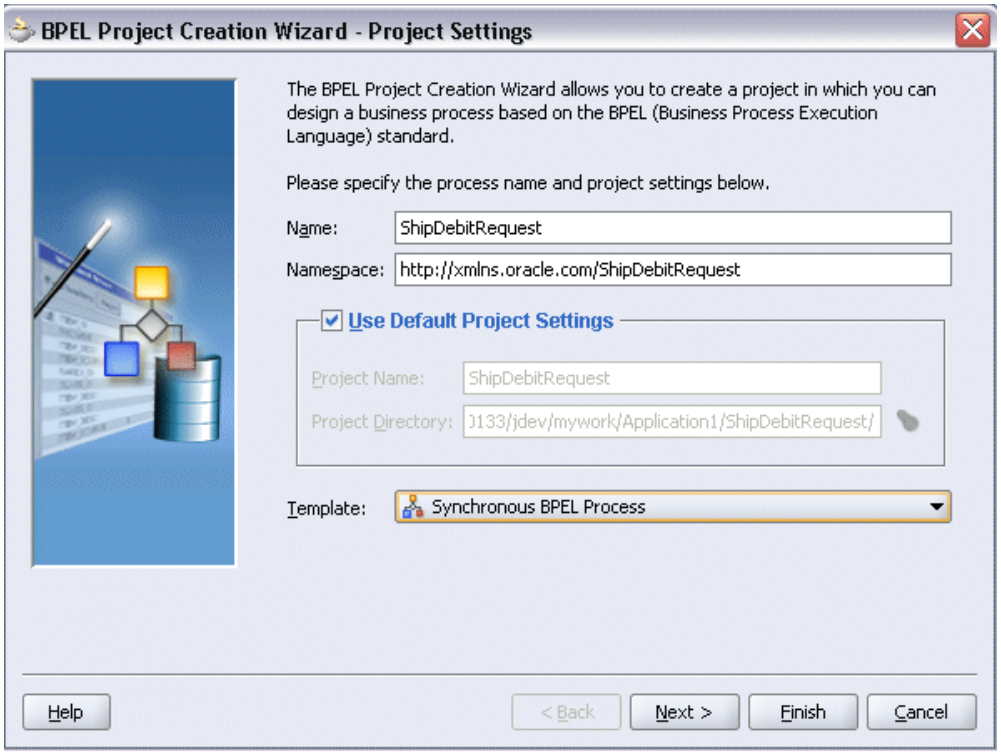
Creating a New BPEL Project

Use this step to create a new BPEL project that will contain various BPEL process activities.

To create a new BPEL project:

1. Open JDeveloper BPEL Designer.
2. From the **File** menu, select **New**. The New Gallery dialog box appears.
3. Select **All Items** from the **Filter By** box. This produces a list of available categories.
4. Expand the **General** node and then select **Projects**.
5. Select **BPEL Process Project** from the **Items** group.
6. Click **OK**. The BPEL Project Creation dialog box appears.

BPEL Project Creation Wizard - Project Settings



7. In the **Name** field, enter a descriptive name such as `ShipDebitRequest`.

   Note: SOA Provider does not support service creation for PL/SQL stored procedures or packages which have '$' character in parameter type names. The presence of $ in the name would cause the XSD generation to fail.

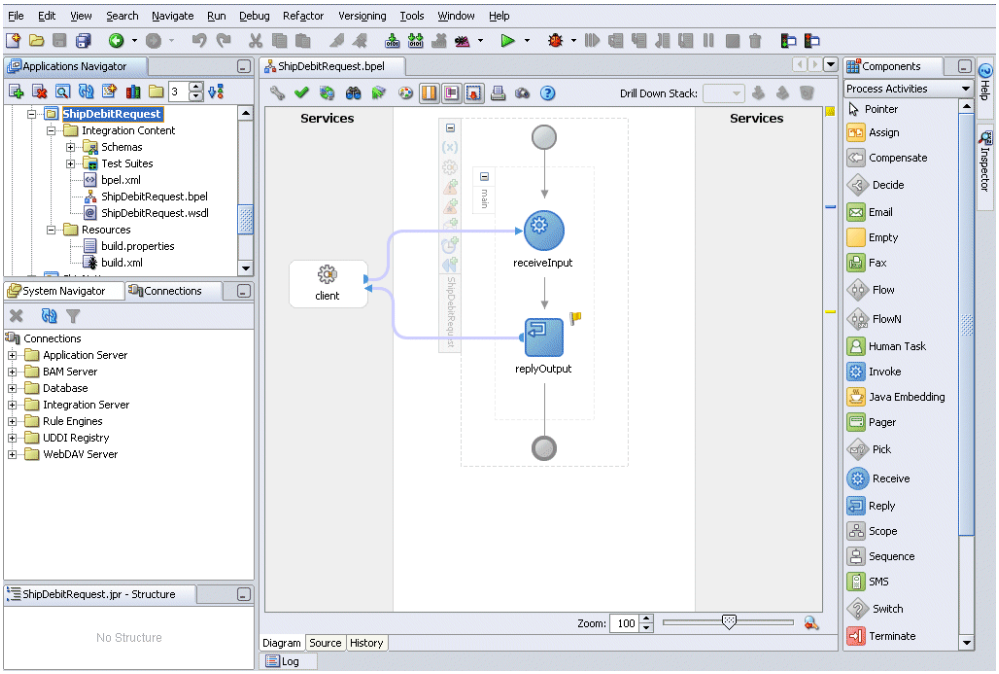
8. From the Template list, select **Synchronous BPEL Process**. Select **Use Default Project Settings**.

9. Use the default input and output schema elements in the Input/Output Elements dialog box.

10. Click **Finish**.

    A new synchronous BPEL process is created with the Receive and Reply activities. The required source files including bpel.xml, using the name you specified (for example `ShipDebitRequest.bpel`) are also generated.

Oracle JDevelopr with BPEL Process Diagram



Creating a Partner Link for the Web Service

Use this step to create a Partner Link called `OZF_SD_REQUEST_PUB`.

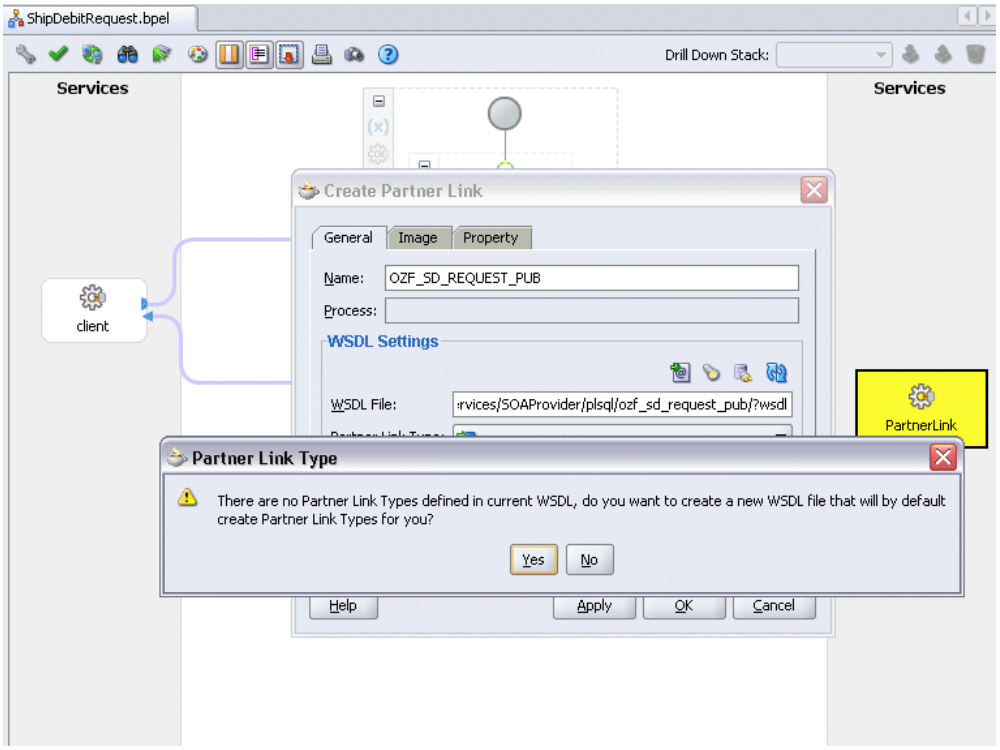
To create a partner link for OZF_SD_REQUEST_PUB Web service:

1. In JDeveloper BPEL Designer, drag and drop the **PartnerLink** service from the Component Palette into the Partner Link border area of the process diagram. The Service Name dialog box appears.

2. Copy the WSDL URL corresponding to the `OZF_SD_REQUEST_PUB` service that you recorded earlier from the Integration Repository, and paste it in the WSDL File field.

3. A Partner Link Type message appears asking whether you want the system to create a new WSDL file that will by default create partner link types for you.

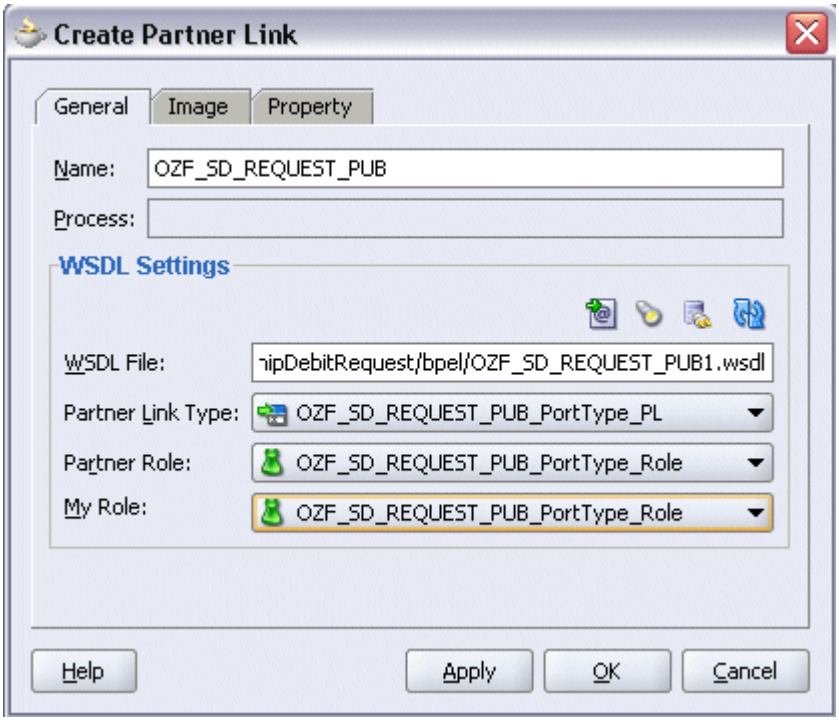
Create Partner Link Dialog with Partner Link Type Pop-Up Message



Click **Yes** to have the Partner Name value populated automatically. The name is defaulted to OZF_SD_REQUEST_PUB.

Select Partner Role and My Role fields from the drop-down lists.

Create Partner Link Dialog



Click **Apply**.

The partner link is created with the required WSDL settings, and is represented in the BPEL project by a new icon in the border area of the process diagram.

4. **Passing Values to Security Headers**

Select the Property tab and click the **Create Property** icon to select the following properties from the property name drop-down list in order to pass the security headers along with the SOAP request:

- wsseUsername

  Specify the username to be passed in the Property Value box.

- wssePassword

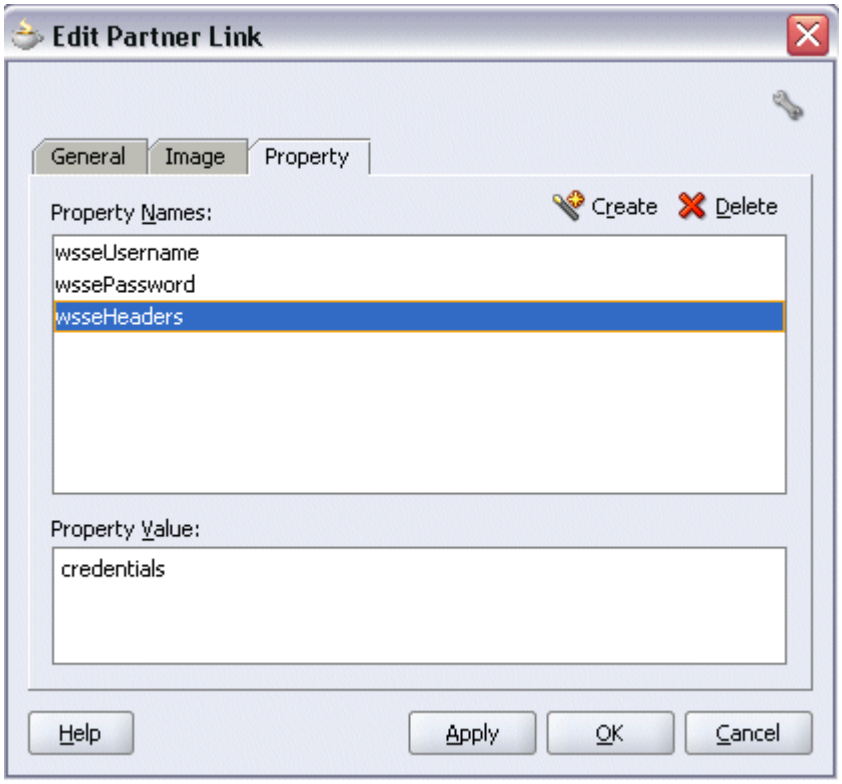
  Specify the corresponding password for the username to be passed in the Property Value box.

- wsseHeaders

  Enter `credentials` as the property value.

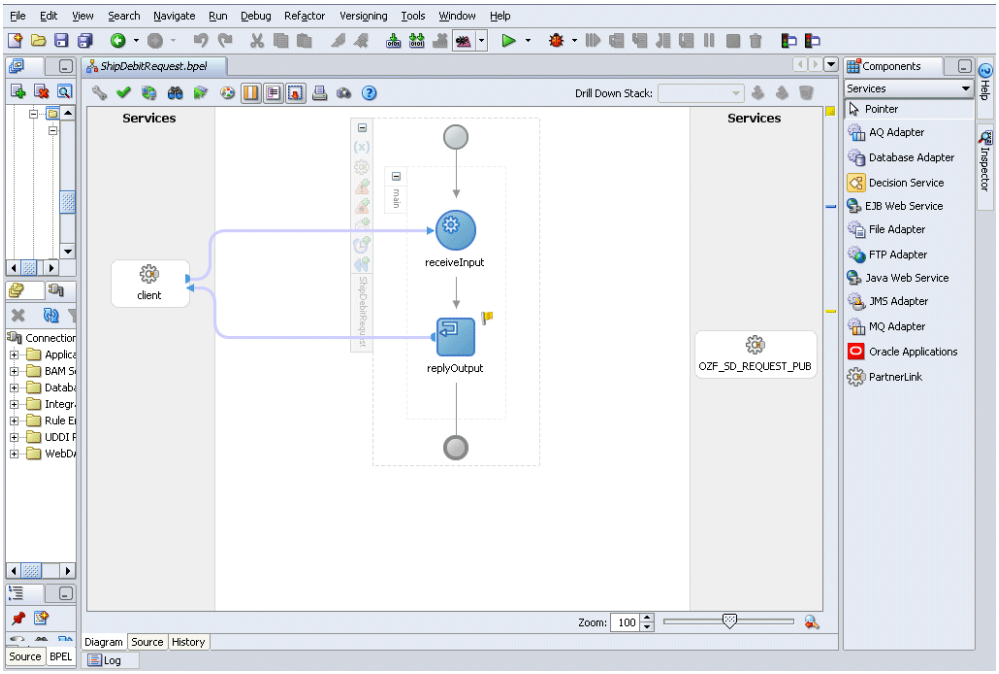
Click **Apply** to save the selected property values.

Edit Partner Link Dialog with Property Tab



5. Click **OK** to complete the partner link configuration.

BPEL Process Diagram with Partner Link Created



Partner Link `OZF_SD_REQUEST_PUB` is added to the Services section in the BPEL process diagram.

Adding a Partner Link for File Adapter

Use this step to configure a BPEL process by reading current contents of a file.

To add a Partner Link for File Adapter to Read Payload:

1. In JDeveloper BPEL Designer, drag and drop the **File Adapter** service from the **Adapter Service** section of the Component Palette into the Partner Link area of the process diagram. The Adapter Configuration wizard welcome page appears.

2. Click **Next**. The Service Name dialog box appears.

3. Enter a name for the file adapter service such as `ReadPayload`. You can add an optional description of the service.

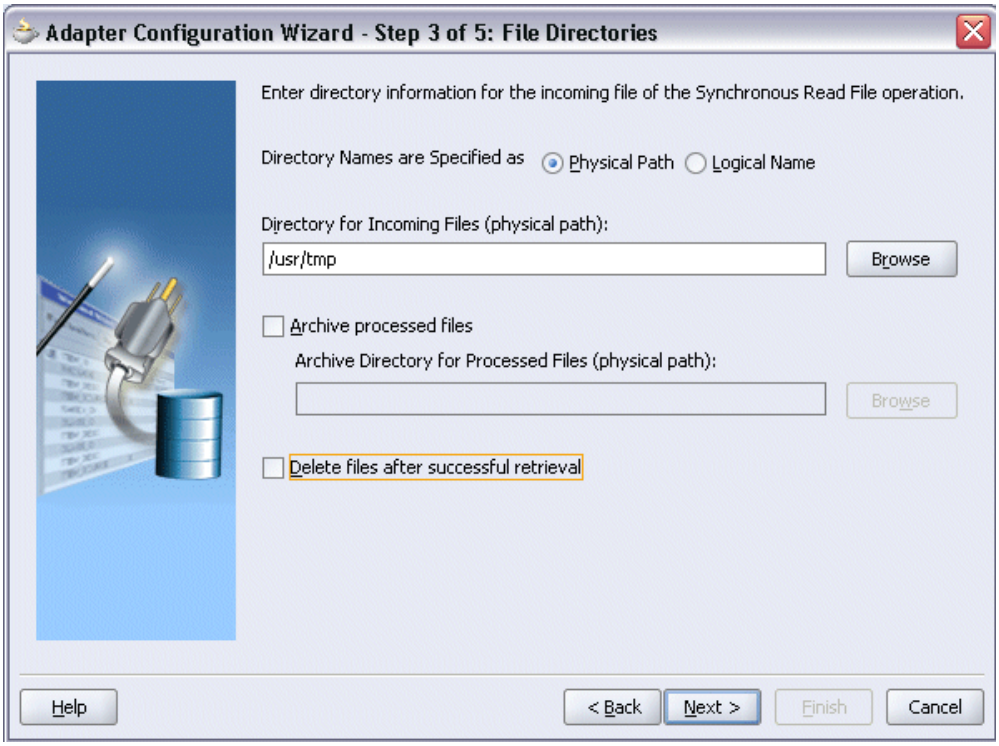
4. Click **Next**. The Operation dialog box appears.

Operation Dialog

5. Specify the operation type, for example **Synchronous Read File**. This automatically populates the **Operation Name** field.

   Click **Next** to access the File Directories dialog box.

File Directories Dialog



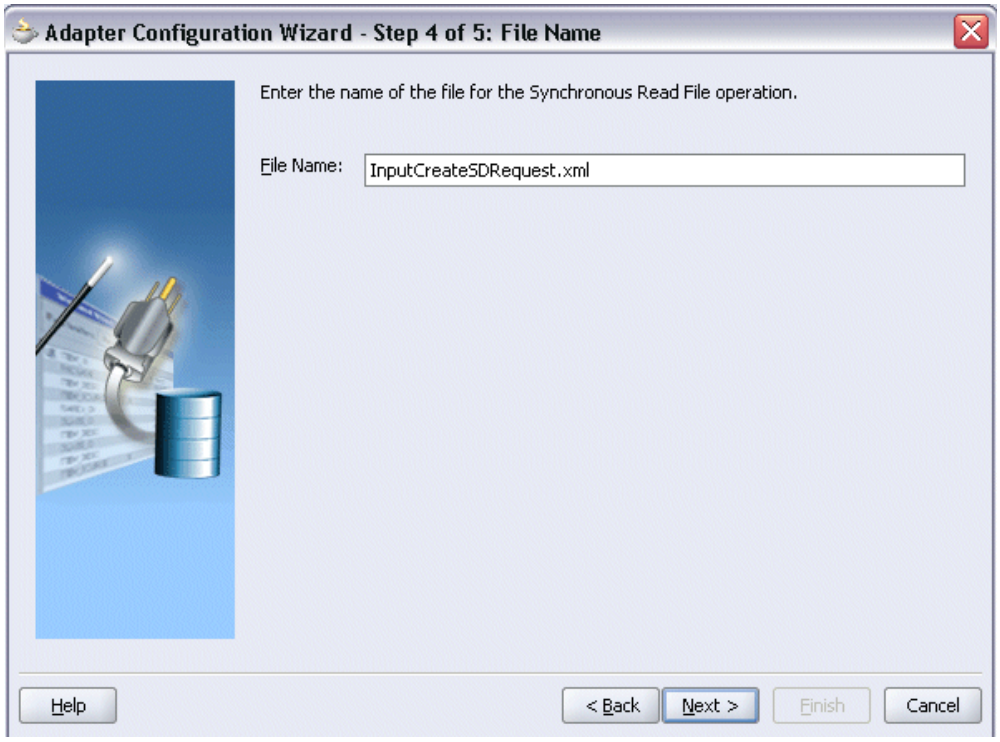
6. Select **Physical Path** radio button and enter the input payload file directory information. For example, enter `/usr/tmp/` as the directory name.

   Note: You must ensure the input payload file `InputCreateSDRequest.xml` is available in the directory '/usr/tmp/' folder of SOA Suite server (or D:\HOL in case of SOA Server in Windows machine).

   Uncheck the **Delete Files after successful retrieval** check box. Click **Next** to open the File Name dialog box.

7. Enter the name of the file for the synchronous read file operation. For example, enter `InputCreateSDRequest.xml`.

File Name Dialog

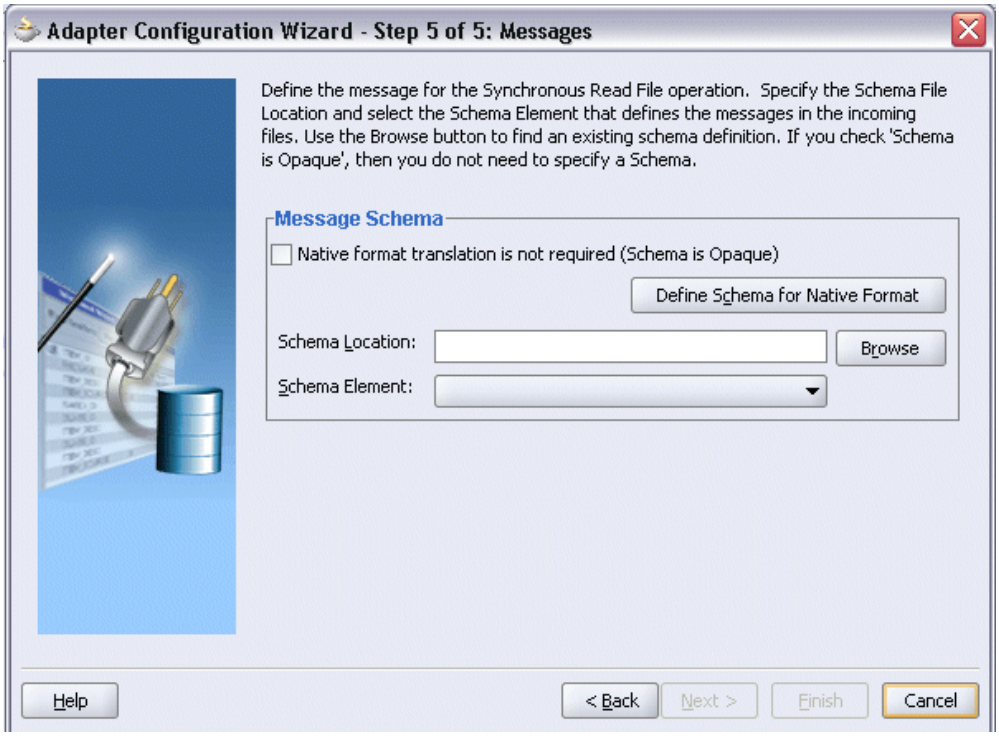


Click **Next**. The Messages dialog box appears.

8. Select **Browse** for schema file in Schema Location.

   The Type Chooser window is displayed.

Messages Dialog



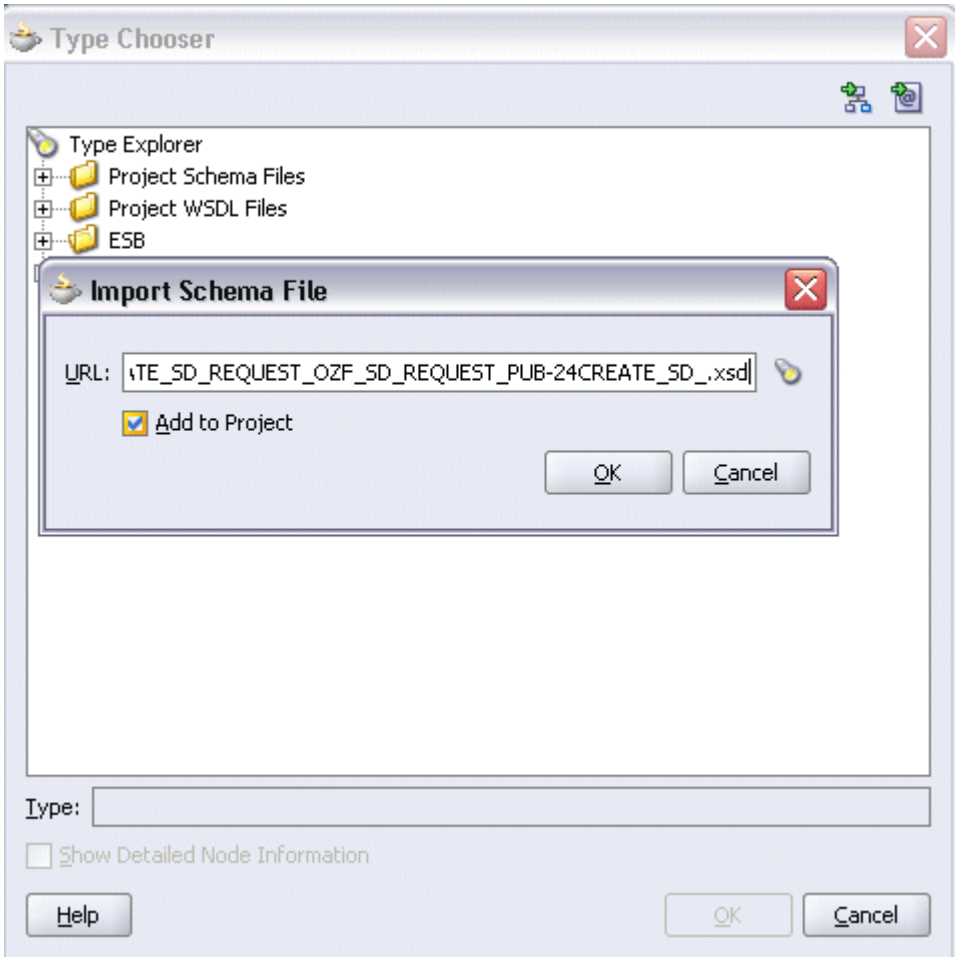
Click **Import Schema Files** button on the top right corner of the Type Chooser window.

Enter the schema location for the service. Such as `http://<myhost>:<port>/webservices/SOAProvider/plsql/ozf_sd_request_pub/APPS_ISG_CREATE_SD_REQUEST_OZF_SD_REQUEST_PUB-24CREATE_SD_.xsd`.

Schema location for your service can be found from the service WSDL URL (for example, `http://<myhost>:<port>/webservices/SOAProvider/plsql/ozf_sd_request_pub/?wsdl`).

Select the **Add to Project** check box and click **OK**.

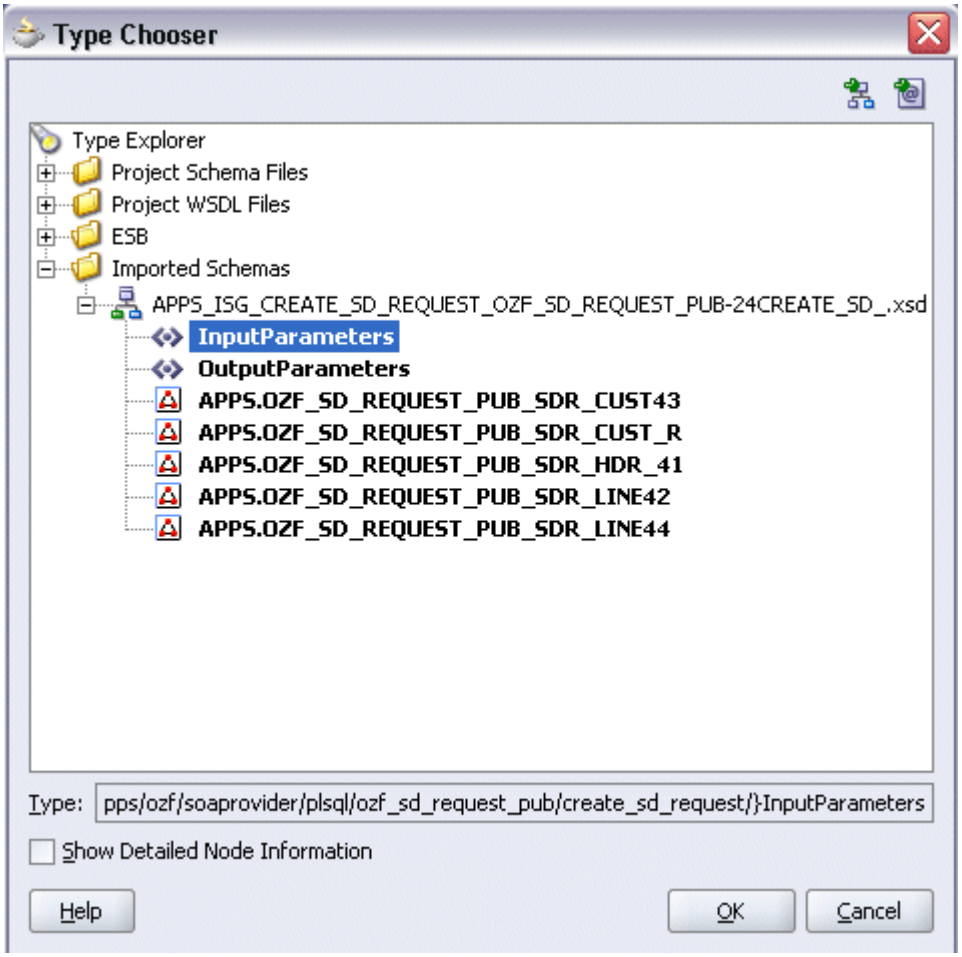
Import Schema File Pop-up Window



Click **OK** for Import schema prompt.

The Imported Schemas folder is automatically added to the Type Chooser window.

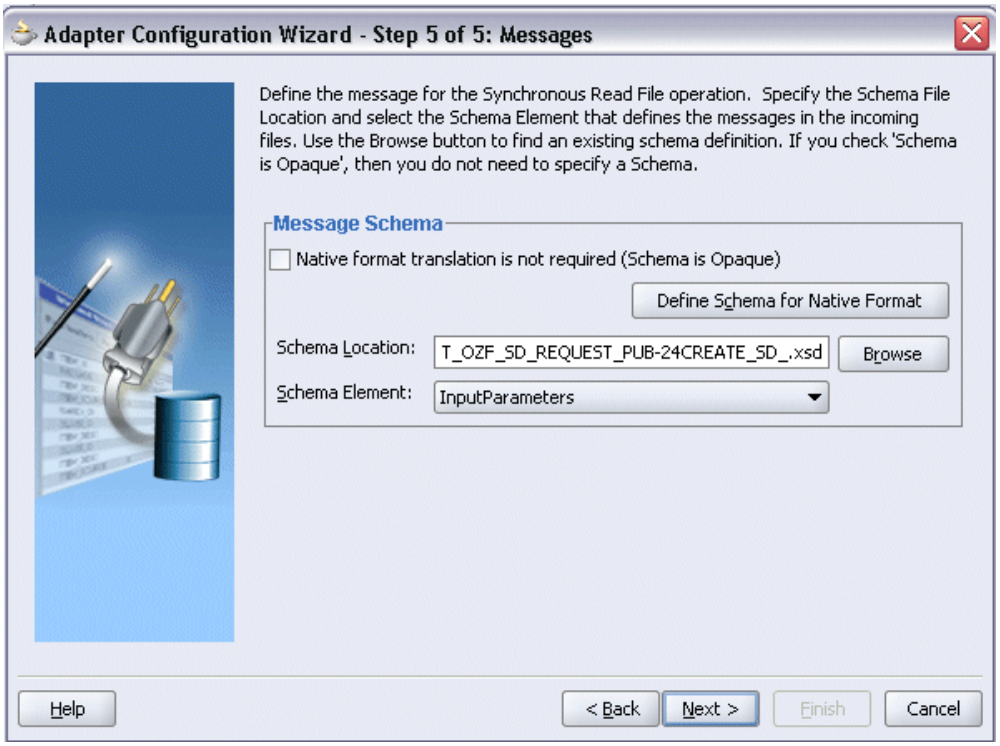
Type Chooser Window with Imported Schema



Expand the Imported Schemas folder and select InputParameters Message in the `APPS_ISG_CREATE_SD_REQUEST_OZF_SD_REQUEST_PUB-24CREATE_SD_.xsd`. Click **OK**.

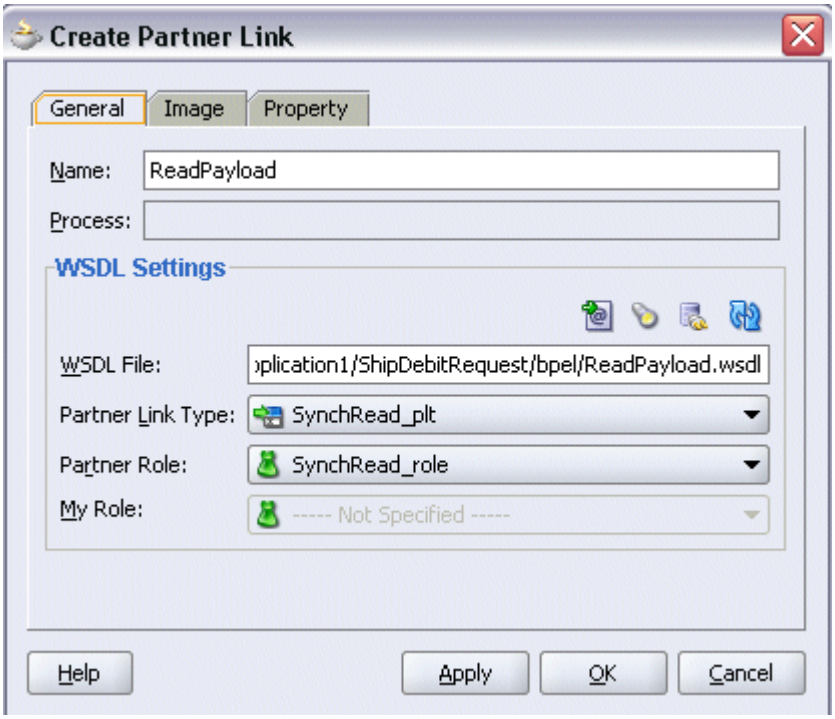
The selected xsd is displayed as Schema Location, and the InputParameters is selected as Schema Element.

Messages Dialog with Selected Schema and Element



9. Click **Next** and then **Finish**. The wizard generates the WSDL file corresponding to the partner link. The main Create Partner Link dialog box appears, specifying the new WSDL file `ReadPayload.wsdl`.

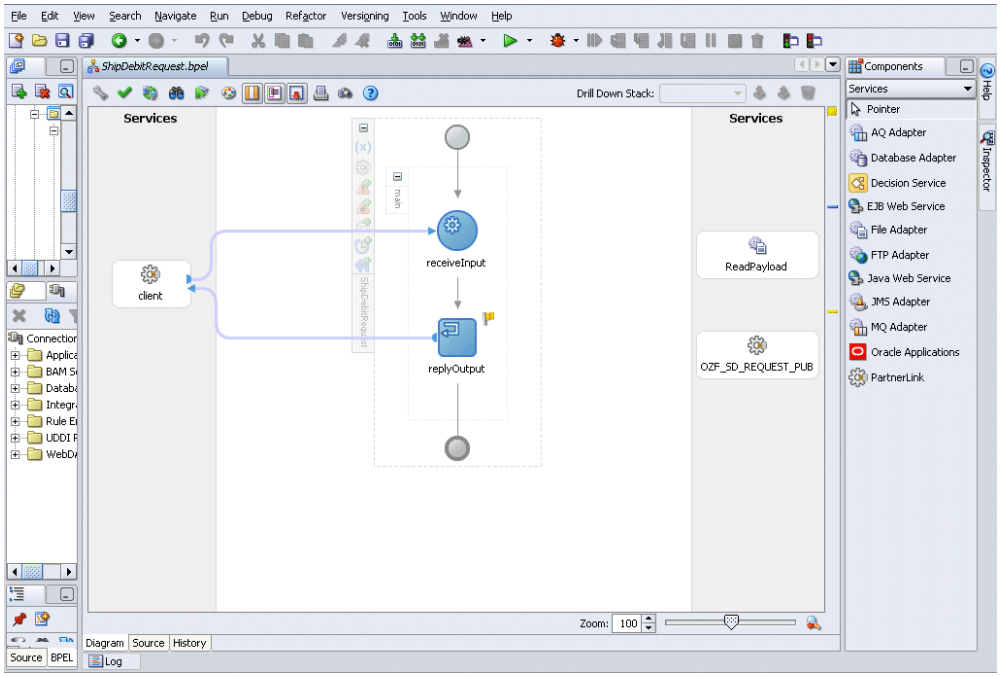
Create Partner Link Dialog with Required Information



Click **Apply** and then **OK** to complete the configuration and create the partner link with the required WSDL settings for the File Adapter Service.

The `ReadPayload` Partner Link appears in the BPEL process diagram.

BPEL Process Dialog with Partner Link for File Adapter



Adding Invoke Activities

This step is to configure two Invoke activities:

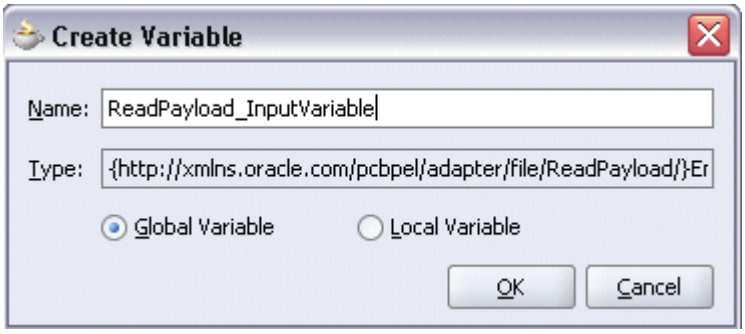
- Read request creation details that is passed from the first Assign activity using `ReadPayload` partner link for File Adapter.
- Send the payload and request details received from the Assign activities to create a ship and debit request by using the `OZF_SD_REQUEST_PUB` partner link.

To add an Invoke activity for ReadPayload Partner Link:

1. In JDeveloper BPEL Designer, drag and drop the **Invoke** activity from the Component Palette into the Activity box of the process diagram, between the **Receive** and **Reply** activities.
2. Link the Invoke activity to the `ReadPayload` service. The Edit Invoke dialog box appears.
3. Enter a name for the Invoke activity, and then click the **Create** icon next to the **Input Variable** field to create a new variable. The Create Variable dialog box appears.

   Enter 'ReadPayload_InputVariable' as the input variable name. You can also accept the default name.

Create Variable Dialog



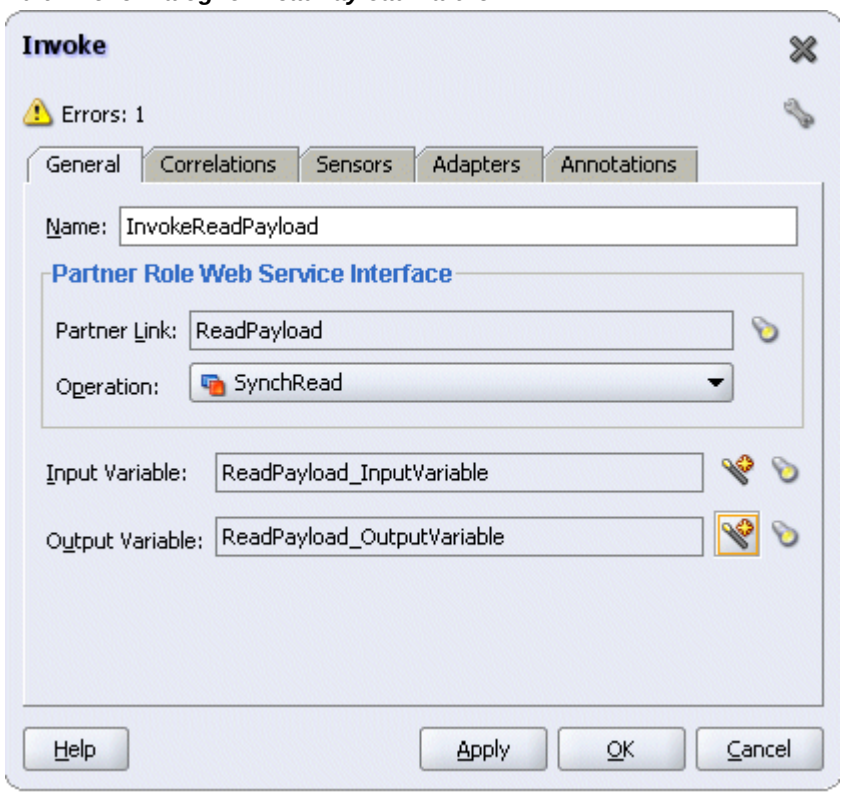
Select **Global Variable**, and then enter a name for the variable. Click **OK**.

4. Click the **Create** icon next to the **Output Variable** field to create a new variable. The Create Variable dialog box appears.

   Enter 'ReadPayload_OutputVariable' as the output variable name. You can also accept the default name.

   Select **Global Variable**, and then enter a name for the variable. Click **OK**.

Edit Invoke Dialog for ReadPayload Partner Link



5. Click **Apply** and then **OK** in the Edit Invoke dialog box to finish configuring the Invoke activity.

   The Invoke activity appears in the process diagram.

To add an Invoke activity for OZF_SD_REQUEST_PUB Partner Link:

1. In JDeveloper BPEL Designer, drag and drop the **Invoke** activity from the Component Palette into the Activity box of the process diagram, after the first **Invoke** activity and the **Reply** activity.

2. Link the Invoke activity to the `OZF_SD_REQUEST_PUB` service. The Edit Invoke dialog box appears.

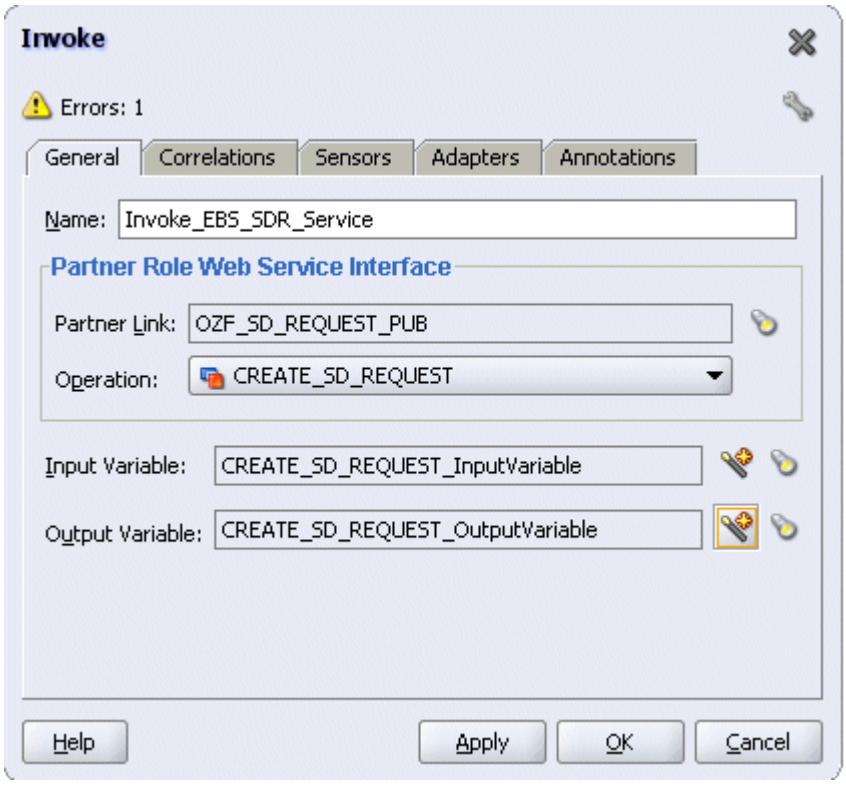
3. Enter a name for the Invoke activity such as 'Invoke_EBS_SDR_Service'.

   In the Operation field, select CREATE_SD_REQUEST from the drop-down list.

4. Create global Input and Output variables as CREATE_SD_REQUEST_InputVariable and CREATE_SD_REQUEST_OutputVariable.

   Click **OK** in Edit Invoke.

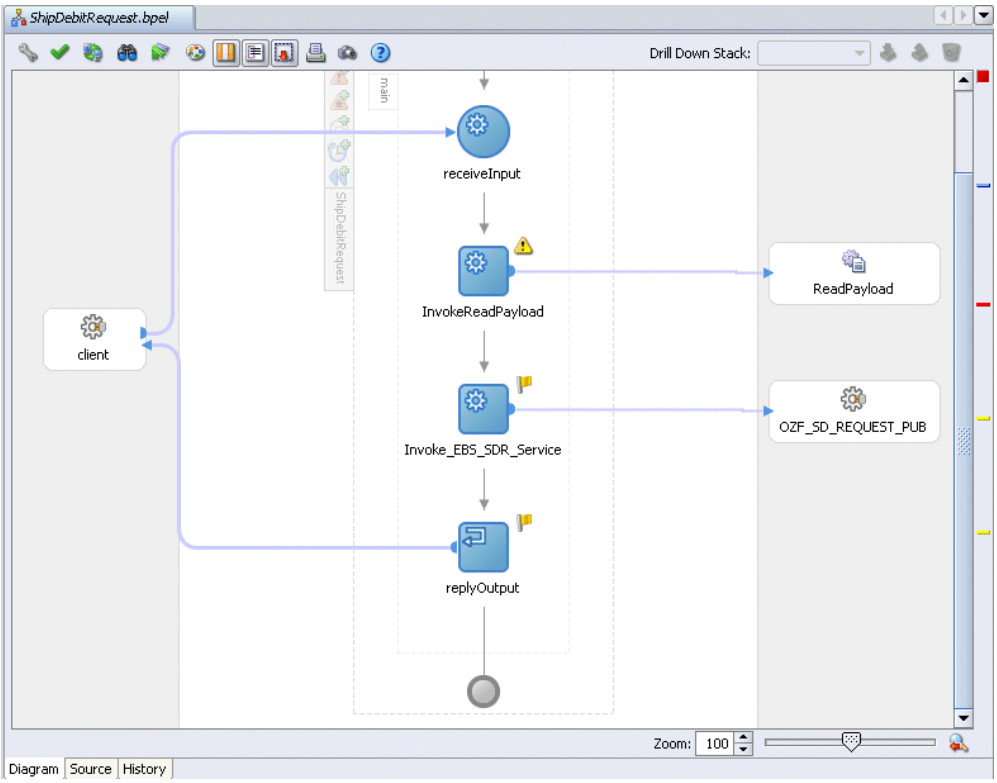
Edit Invoke Dialog for OZF_SD_REQUEST_PUB Partner Link



Click **Apply** and then **OK** in the Edit Invoke dialog box to finish configuring the Invoke activity.

The Invoke activity appears in the process diagram.

BPEL Process Diagram with Two Invoke Activities



Adding Assign Activities

This step is to configure four Assign activities:

1. To set the SOAHeader details for ship and debit SOAP request.

   Note: You also need to populate certain variables in the BPEL process for SOAHeader elements to pass values that may be required to set application context during service execution. These SOAHeader elements are *`Responsibility`*, *`RespApplication`*, *`SecurityGroup`*, *`NLSLanguage`*, and *`Org_Id`*.

2. To set input payload for SOAP request.

3. To set input for SOAP request.

4. To set the SOAP response to output.

To add the first Assign activity to set SOAHeader details:

Assigning SOAHeader Parameters:

1. In JDeveloper BPEL Designer, drag and drop the **Assign** activity from the Component Palette into the Activity box of the process diagram, between two **Invoke** activities.

2. Double-click the **Assign** activity to access the Edit Assign dialog box.

3. Click the General tab to enter the name for the Assign activity, such as 'SetSOAHeader'.

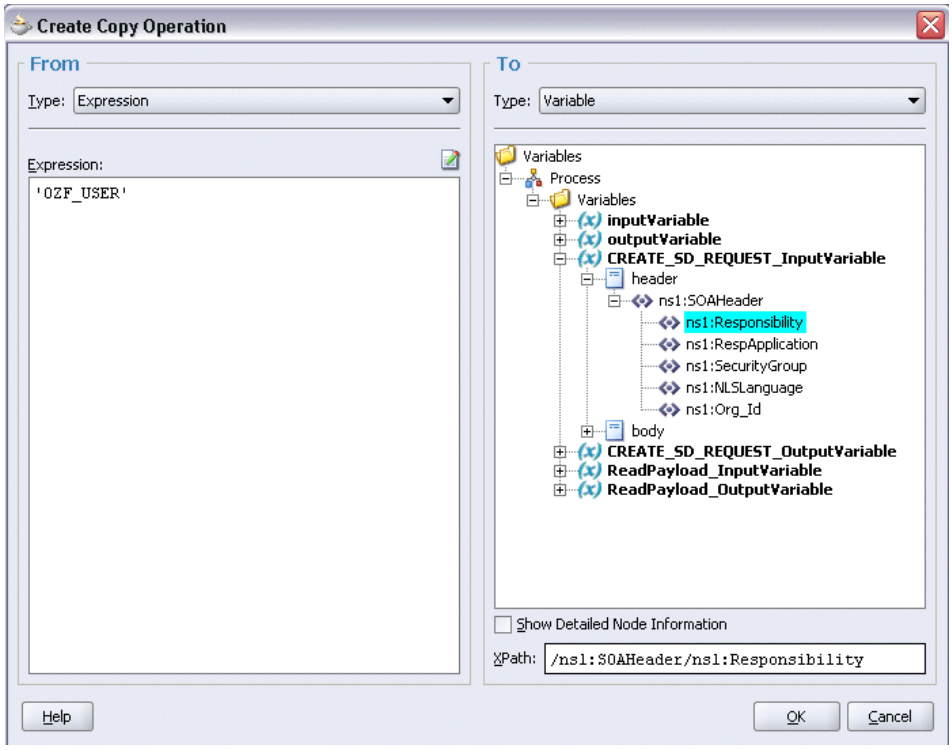
4. On the Copy Operation tab, click **Create** and then select **Copy Operation** from the menu. The Create Copy Operation window appears.

5. Enter the first pair of parameters:

   - In the From navigation tree, select type Expression and then enter 'OZF_USER' in the Expression box.

   - In the To navigation tree, select type Variable. Navigate to **Variables > Process > Variables > CREATE_SD_REQUEST_InputVariable > header > ns1: SOAHeader** and select **ns1:Responsibility**. The XPath field should contain your selected entry.

Create Copy Operation Dialog to Assign Responsibility Parameter



- Click **OK**.

6. Enter the second pair of parameters by selecting **Copy Operation** from the Create drop-down list with the following values:

   - In the From navigation tree, select type Expression and then enter 'OZF' in the Expression box.

   - In the To navigation tree, select type Variable. Navigate to **Variables > Process > Variables > CREATE_SD_REQUEST_InputVariable > header > ns1: SOAHeader** and select **ns1:RespApplication**. The XPath field should contain your selected entry.

   - Click **OK**.

7. Enter the third pair of parameters:

   - In the From navigation tree, select type Expression and then enter 'STANDARD' in the Expression box.

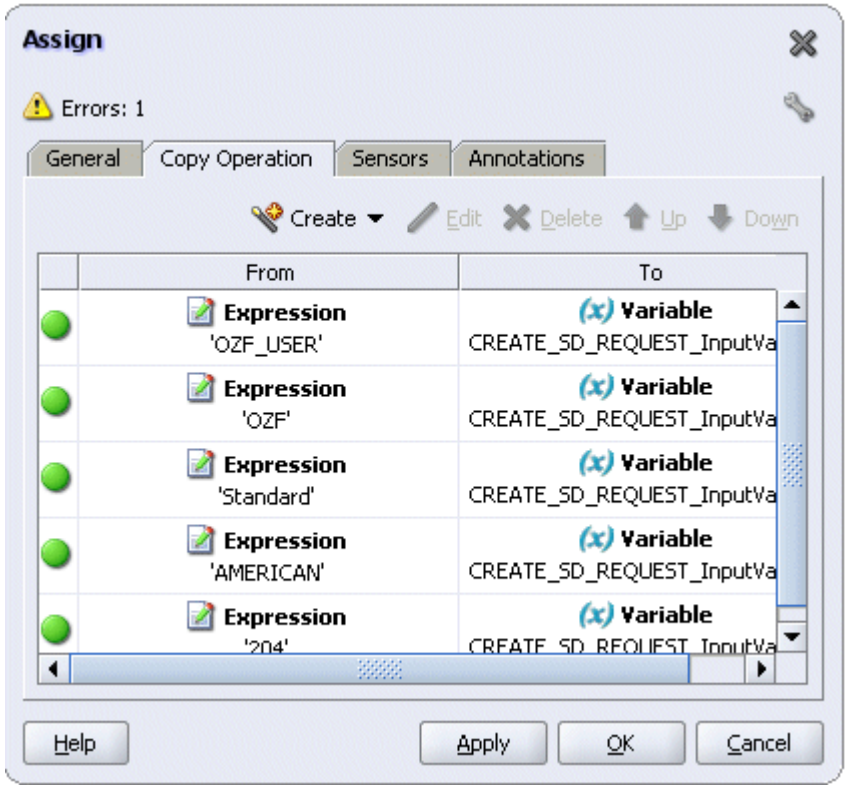
   - In the To navigation tree, select type Variable. Navigate to **Variables > Process > Variables > CREATE_SD_REQUEST_InputVariable > header > ns1:**

SOAHeader and select **ns5:SecurityGroup**. The XPath field should contain your selected entry.

- Click **OK**.

8. Enter the fourth pair of parameters:
   - In the From navigation tree, select type Expression and then enter 'AMERICAN' in the Expression box.
   - In the To navigation tree, select type Variable. Navigate to **Variables > Process > Variables > CREATE_SD_REQUEST_InputVariable > header > ns1: SOAHeader** and select **ns1:NLSLanguage**. The XPath field should contain your selected entry.
   - Click **OK**.
9. Enter the fifth pair of parameters by selecting **Copy Operation** from the Create drop-down list with the following values:
   - In the From navigation tree, select type Expression and then enter '204' in the Expression box.
   - In the To navigation tree, select type Variable. Navigate to **Variables > Process > Variables > CREATE_SD_REQUEST_InputVariable > header > ns1: SOAHeader** and select **ns1:Org_Id**. The XPath field should contain your selected entry.
   - Click **OK**.
10. The Edit Assign dialog box appears.

Edit Assign Dialog with Copy Operation Tab



11. Click **Apply** and then **OK** to complete the configuration of the Assign activity.

To enter the second Assign activity to pass payload information to the Invoke_EBS_SDR_Service Invoke activity:

1. Add the second Assign activity by dragging and dropping the **Assign** activity from the Component Palette into the Activity box of the process diagram, between the 'SetSOAHeader' **Assign** activity and the 'Invoke_EBS_SDR_Service' **Invoke** activity.

2. Repeat Step 2 to Step 4 described in creating the first Assign activity to add the second Assign activity called 'SetPayload'.

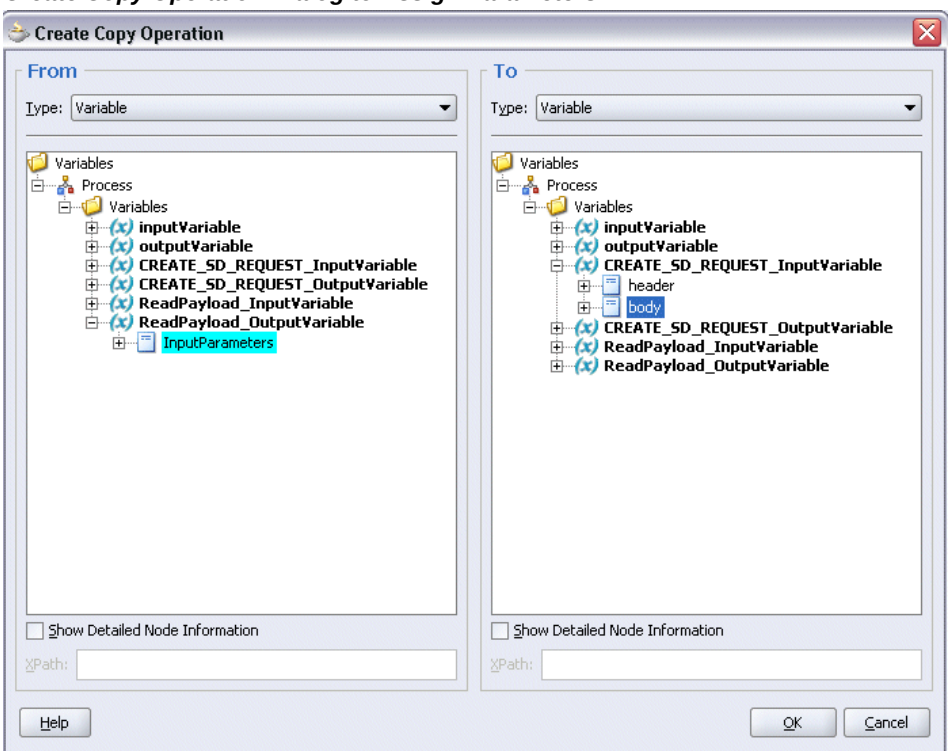
3. Enter the following information:

   - In the From navigation tree, navigate to **Variable > Process > Variables > ReadPayload_OutVariable** and select **InputParameters**.

   - In the To navigation tree, select type Variable and then navigate to **Variable > Process > Variables > CREATE_SD_REQUEST_InputVariable** and select **Body**.

- Click **OK**.

Create Copy Operation Dialog to Assign Parameters



4. The Edit Assign dialog box appears.

5. Click **Apply** and then **OK** to complete the configuration of the Assign activity.

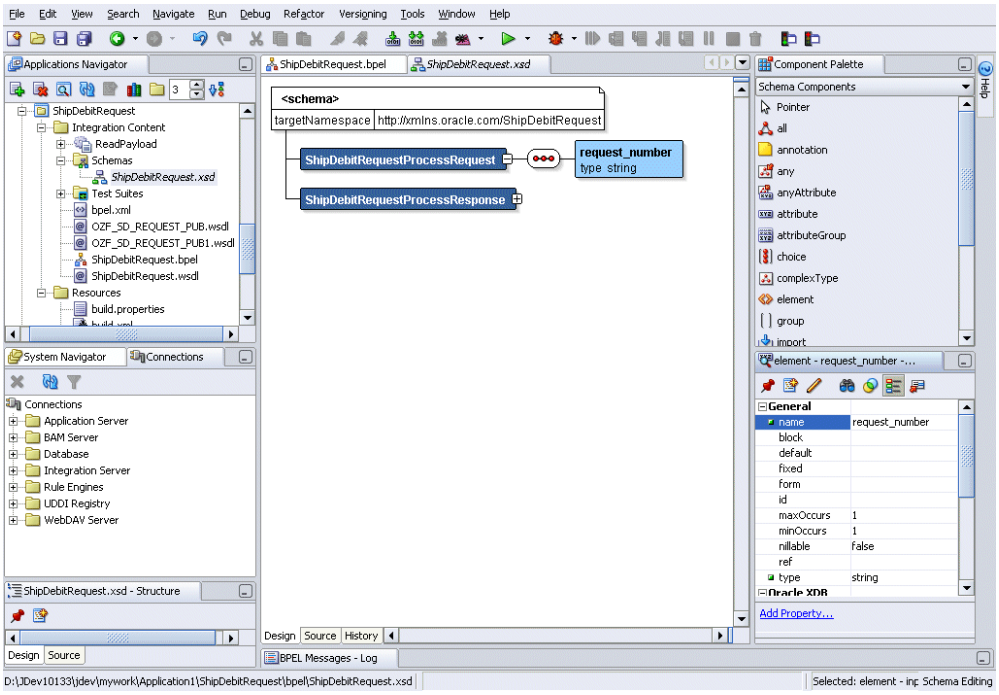
Defining Schema for BPEL Process Input Request

Before setting the input request for the SOAP request, you need to define necessary schema for BPEL process request.

1. From the Applications Navigator window, expand the **ShipDebitRequest > Integration Content > Schemas** folder to open the `ShipDebitRequest.xsd` file.

2. In the Design mode, expand 'ShipDebitRequestProcessRequest' to view elements within process request.

Schema Creation for BPEL Process Request



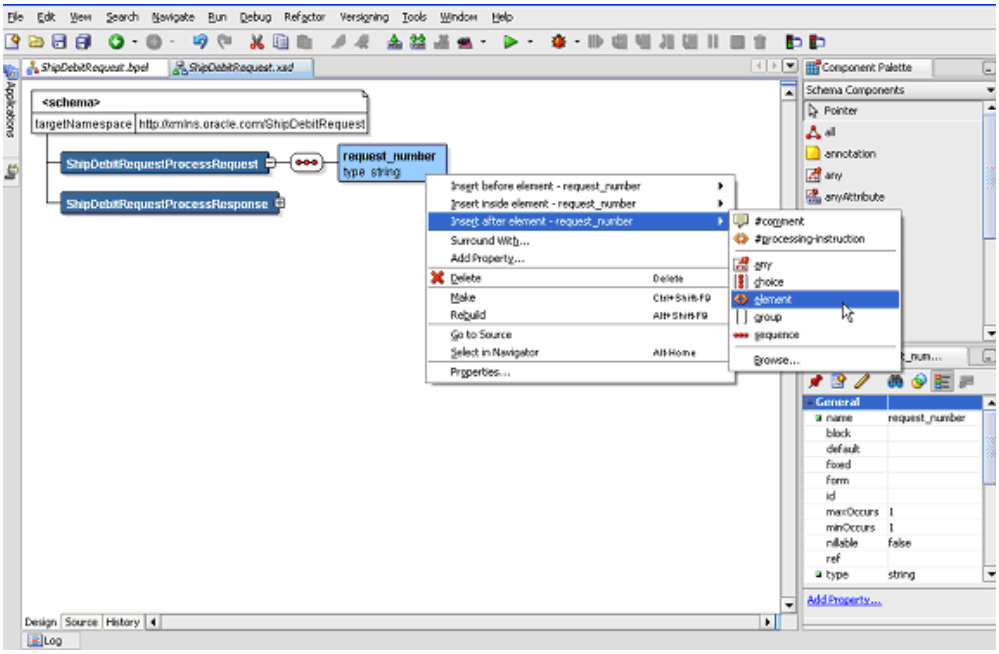
3. From element properties, change the name from 'input' to 'request_number'.

4. Select and right-click on the 'request_number' element to open the pop-up menu.

   Select **Insert after element – request_number > element** option. New element 'element1' is displayed in the schema design window underneath the 'request_number' element.

   From element properties section, change the name from 'element1' to 'description' and enter type as 'string'.

Schema Elements Creation

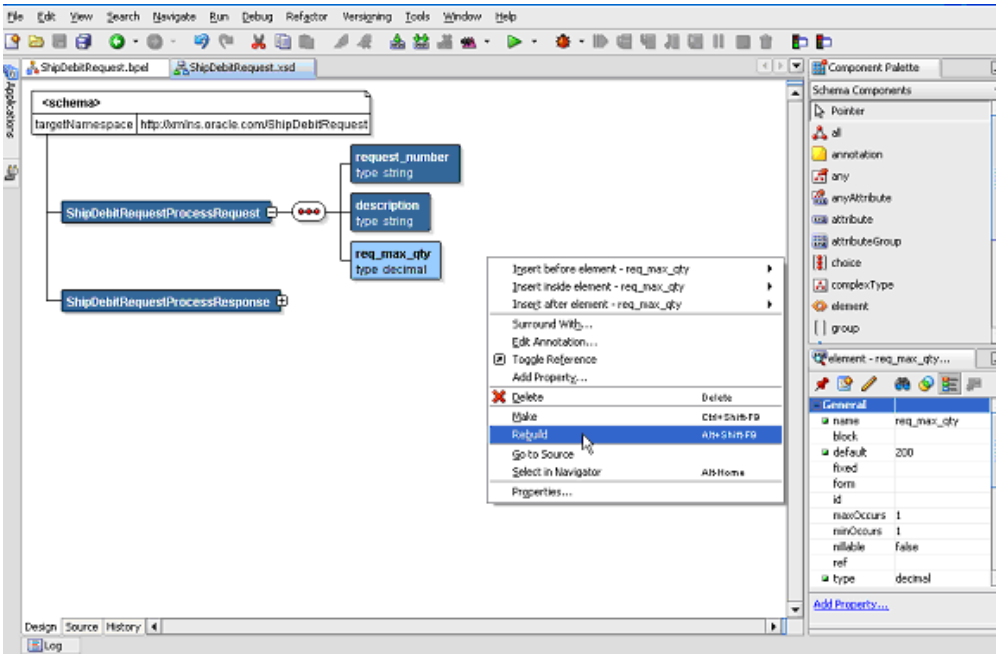


5. Similarly insert another element called 'req_max_qty' after element 'description'.

   Enter default value as '200' and type as 'decimal'.

   Right-click on mouse and select **Rebuild** option.

Schema Elements After the Rebuild



Look for compilation messages in Log to ensure the successful compilation.

To set the third Assign activity to pass the input request to the Invoke_EBS_SDR_Service Invoke activity:

1. Add the third Assign activity by dragging and dropping the **Assign** activity from the Component Palette into the Activity box of the process diagram, between the second **Assign** activity 'SetPayload' and the `Invoke_EBS_SDR_Service` **Invoke** activity.

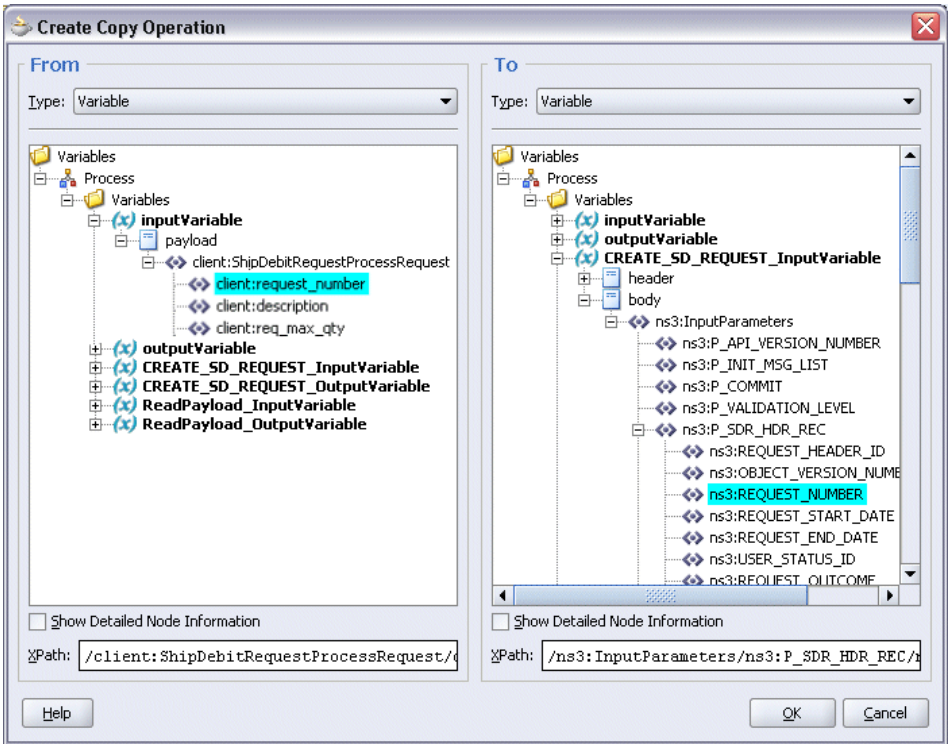
2. Repeat Step 2 to Step 4 described in creating the first Assign activity to add the third Assign activity called 'SetInput'.

3. Enter the following information:

   - In the From navigation tree, navigate to **Variable > Process > Variables > inputVariable > Payload > client:ShipDebitRequestProcessRequst** and select **client:request_number**. The XPath field should contain your selected entry.

   - In the To navigation tree, select type Variable and then navigate to **Variable > Process > Variables > Create_SD_REQUEST_InputVariable > Body > ns3: InputParameters >ns3:P_SDR_HDR_REC** and select **ns3: REQUEST_NUMBER**. The XPath field should contain your selected entry.

Create Copy Operation Dialog for the Third Assign Activity



- Click **OK**.

4. Enter the second pair of parameters by selecting **Copy Operation** from the Create drop-down list with the following values:

   - In the From navigation tree, navigate to **Variable > Process > Variables > inputVariable > Payload > client:ShipDebitRequestProcessRequst** and select **client:description**. The XPath field should contain your selected entry.

   - In the To navigation tree, select type Variable and then navigate to **Variable > Process > Variables > Create_SD_REQUEST_InputVariable > Body > ns3: InputParameters >ns3:P_SDR_HDR_REC** and select **ns3: REQUEST_DESCRIPTION**. The XPath field should contain your selected entry.

   - Click **OK**.

5. Enter the third pair of parameters by selecting **Copy Operation** from the Create drop-down list with the following values:

   - In the From navigation tree, navigate to **Variable > Process > Variables >**

inputVariable > Payload > client:ShipDebitRequestProcessRequst and select **client:req_max_qty**. The XPath field should contain your selected entry.

- In the To navigation tree, select type Variable and then navigate to **Variable > Process > Variables > Create_SD_REQUEST_InputVariable > Body > ns3: InputParameters >ns3:P_SDR_LINES_TBL > ns3:P_SDR_LINES_TBL_ITEM** and select **ns3:MAX_QTY**. The XPath field should contain your selected entry.

- Click **OK**.

6. The Edit Assign dialog box appears.

   Click **Apply** and then **OK** to complete the configuration of the Assign activity.

To add the fourth Assign activity to set SOAP response to output:

1. Add the third Assign activity by dragging and dropping the **Assign** activity from the Component Palette into the Activity box of the process diagram, between the `Invoke_EBS_SDR_Service` **Invoke** and the **Reply** activities.

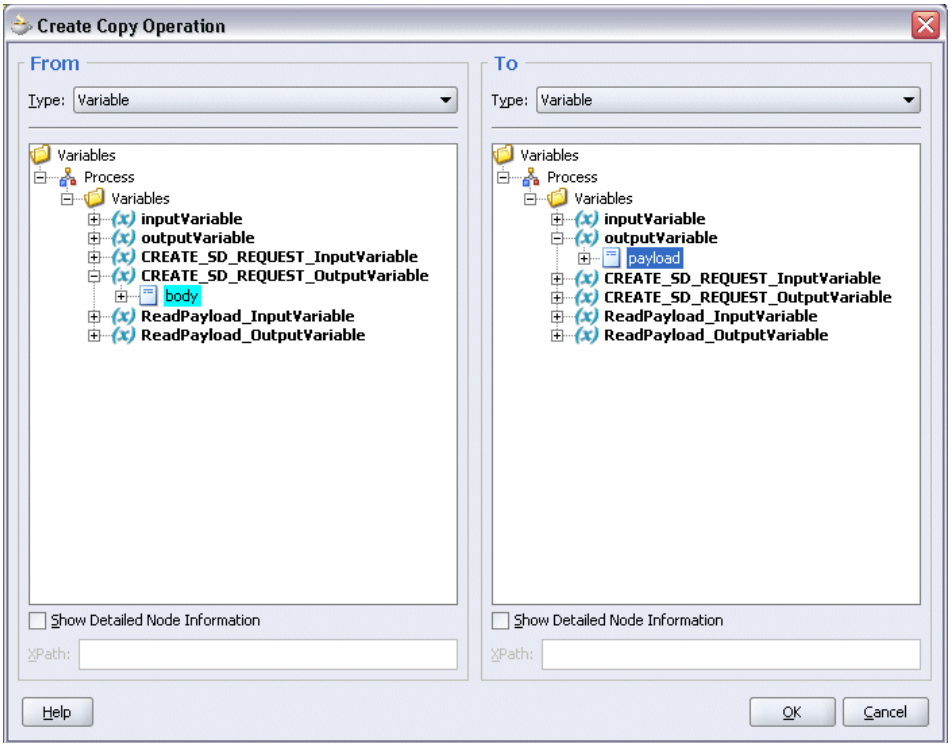
2. Repeat Step 2 to Step 4 described in creating the first Assign activity to add the fourth Assign activity called 'SetResponse'.

3. Enter the following information:

   - In the From navigation tree, select type Variable. Navigate to **Variable > Process > Variables > CREATE_SD_REQUEST_OutputVariable** and select **body**.

   - In the To navigation tree, select type Variable. Navigate to **Variable > Process > Variables > outputVariable** and select **payload**.

Create Copy Operation for the Fourth Assign Activity



- Click **OK**.

4. The Edit Assign dialog box appears.

   Click **Apply** and then **OK** to complete the configuration of the Assign activity.

Deploying and Testing the BPEL Process

After creating a BPEL process using the WSDL URL generated from a PL/SQL interface definition, you can deploy it to a BPEL server if needed. To ensure that this process is modified or orchestrated appropriately, you can also manually test the BPEL process by initiating the business process contained in the BPEL process to test the interface integration.

Prerequisites

Before deploying the BPEL process using Oracle JDeveloper, you must ensure that you have established the connectivity between the deign-time environment and the run-time servers including the application server and the integration server.

How to configure the necessary server connection, see Configuring Server Connection, page B-1.

The payload information for the creation of supplier ship and debit request, see Sample Payload for Creating Supplier Ship and Debit Request, page C-1.

To validate your BPEL process, perform the following run-time tasks:

1. Deploy the BPEL process, page 3-36

   Once you deploy the process to a BPEL server, it becomes available so that you can run the process manually to test it for validation.

2. Test the BPEL process, page 3-37

   After deploying a BPEL process, you can manage the process from the BPEL console to manually initiate the business process and test the interface integration contained in your BPEL process.

Deploying the BPEL Process

You must deploy the BPEL process (`ShipDebitRequest.bpel`) that you created earlier before you can run it.

To deploy the BPEL process:

1. In the Applications Navigator of JDeveloper BPEL Designer, select the **ShipDebitRequest** project.

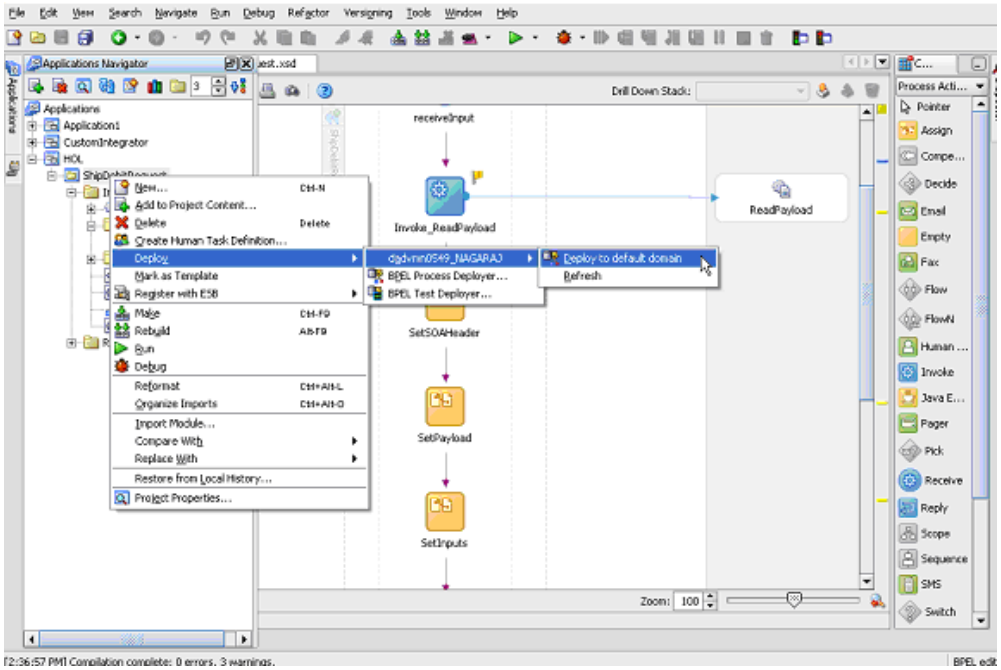
2. Right-click the project and click **Make** action from the menu.

   Look for any compilation error messages in Message Log.

   Right-click the project and select **Deploy >*Integration Server Connection name*> Deploy to Default Domain** action from the menu.

   For example, you can select **Deploy > BPELServerConn > Deploy to Default Domain** to deploy the process if you have the BPEL Process Manager setup appropriately.

Oracle JDeveloper Page to Deploy a SOA Composite with BPEL Process



Look for 'Build successful' message in Apache Ant – Log to ensure that the BPEL project is compiled and successfully deployed.

Testing the BPEL Process

To validate whether the BPEL process that you created works or not, you need to manually initiate the process after it has been successfully deployed to the BPEL server. Therefore, the validation starts with the BPEL console to ensure that you can find the deployed BPEL process listed in the console. Then, you can log on to Oracle E-Business Suite to manually initiate the purchase order approval and acknowledgement processes and to confirm that the relevant event is raised and the updated purchased order details is also written in the XML file.

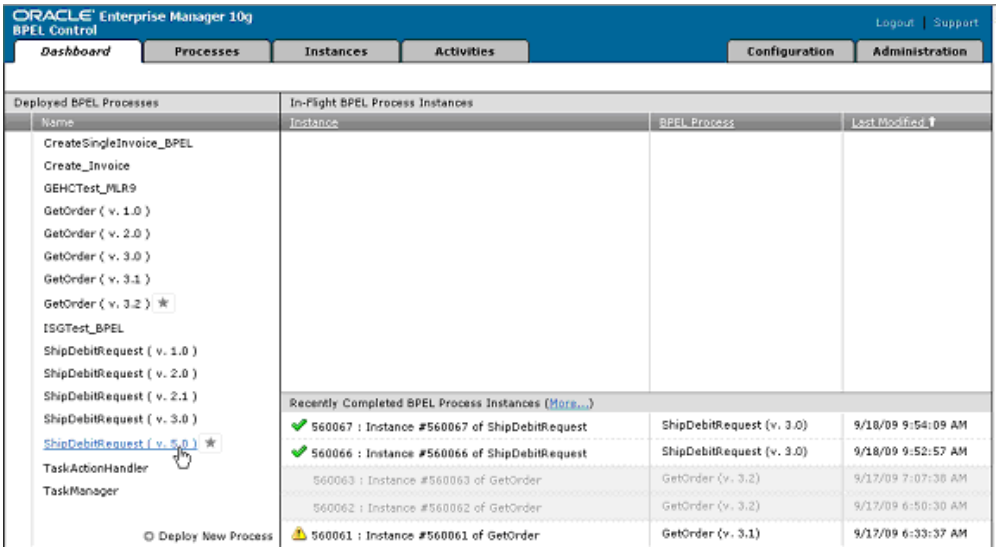
To test the BPEL process:

1. Log in to Oracle Application Server 10g BPEL Console (`http://<soaSuiteServerHostName>:<port>/BPELConsole`). The BPEL Console login page appears.

2. Enter the username and password and click **Login**.

3. In the BPEL Control Console, confirm that ShipDebitRequest has been deployed.

Oracle Enterprise Manager BPEL Control Console to View the Deployed Service



4. Click the ShipDebitRequest link to open the Initiate tab

5. In the payload region, enter the following fields:

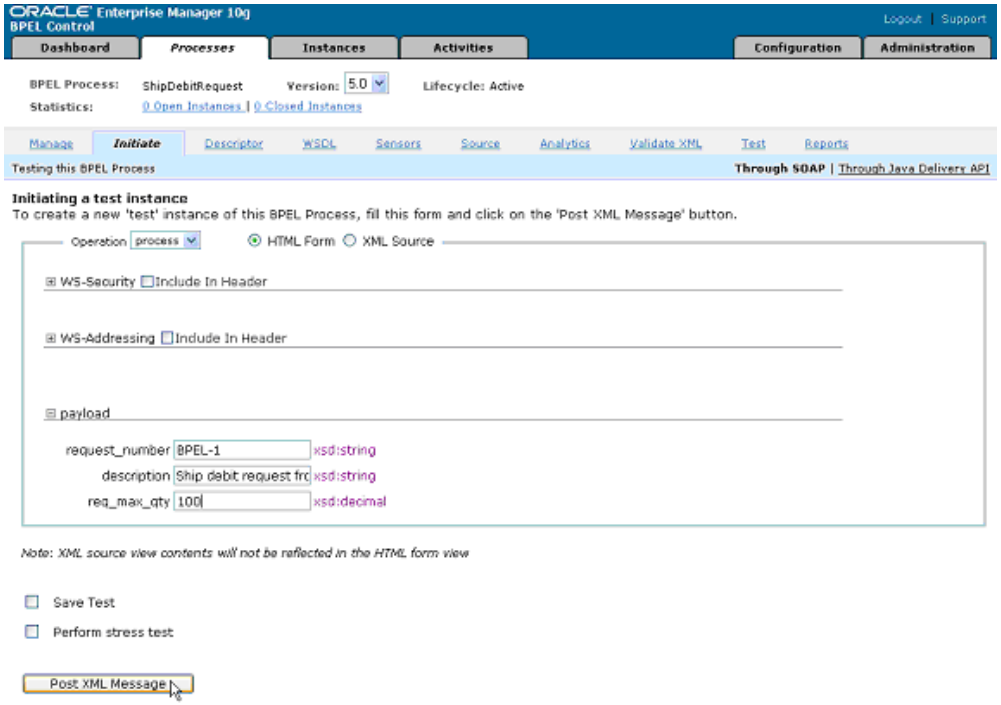
   - request_number: Enter a unique number in this field, such as `BPEL-1`.

     Note: The Request Number entered here should be unique each time that you initiate. The Supplier Ship and Debit Request Number should be unique across users in Supplier Ship and Debit of Oracle Trade Management.

   - description: Enter appropriate description information.

   - req_max_qty: Enter `100` as the value.

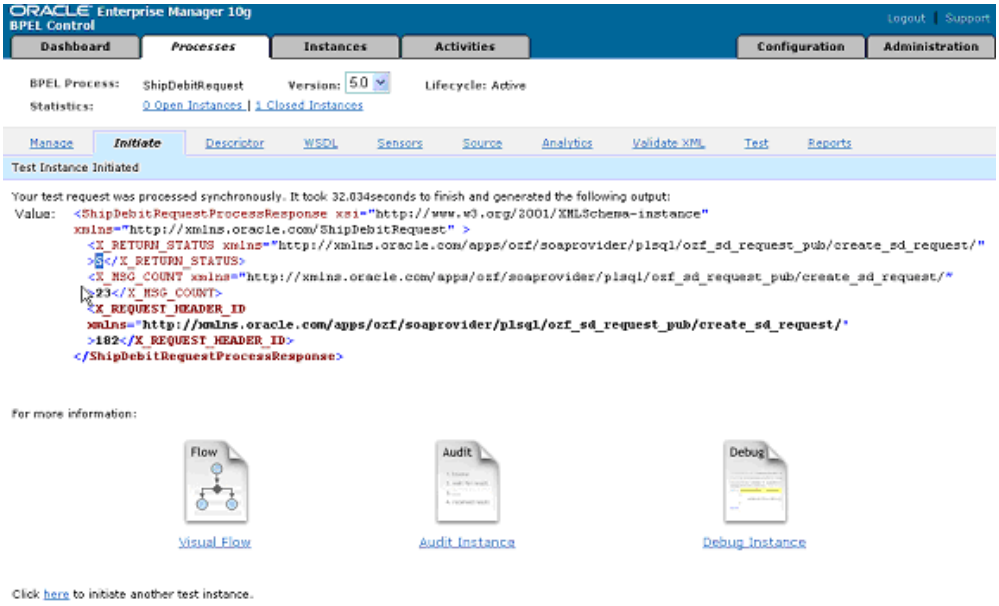
Initiate Tab to Specify Input Payload



Click **Post XML Message** to initiate the process.

6. Verifying SOAP Response in BPEL Console

Initiate Tab to Verify SOAP Response

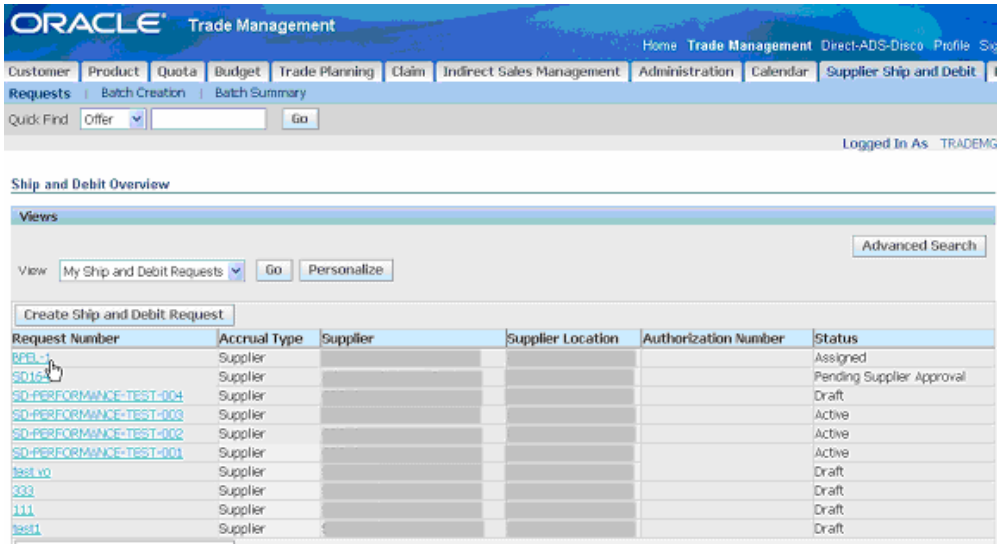


You can view the SOAP response displayed synchronously in BPEL Console. Look for 'S' in X_RETURN_STATUS (for success). If 'E' is displayed in X_RETURN_STATUS, then it means error has occurred while processing the service. Look for detailed exception message in SOA Monitor.

7. **Verifying Created Supplier Ship and Debit Request in Oracle Trade Management**

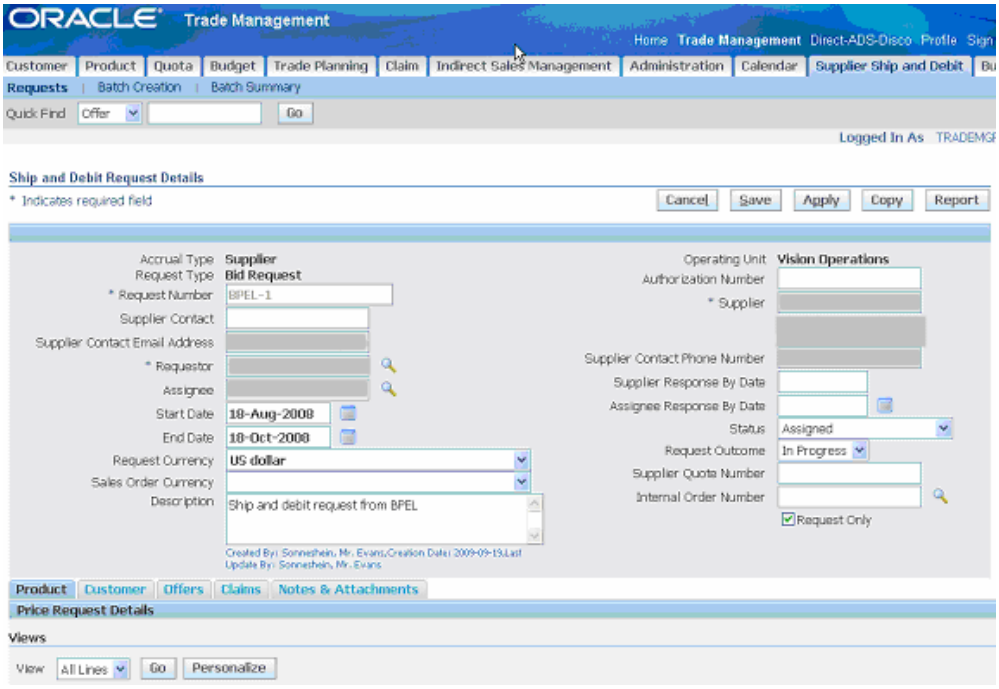
   Log in to Oracle E-Business Suite as a user who has the Oracle Trade Management User responsibility. Select the **Supplier Ship and Debit** link from the navigation menu to open the Ship and Debit Overview window.

Ship and Debit Overview Window



8. Notice that the Request Number `BPEL-1` entered earlier is displayed in the list. Click the request number `BPEL-1` link to open the Ship and Debit Request Details window for the created request. Verify the details.

Ship and Debit Request Details Window to Verify the Ship and Debit Request Details



Using PL/SQL REST Services

REST services provided through Oracle E-Business Suite Integrated SOA Gateway can be used to create or update resources in Oracle E-Business Suite.

Since all REST services are secured by HTTP Basic Authentication or Token Based Authentication, to better understand how these security methods work in conjunction with REST service invocation, the following REST service invocation examples are described in this section:

- Invoking a REST Service Using HTTP Basic Authentication and XML Payload With REST Header, page 3-42
- Invoking a REST Service Using Token Based Authentication and JSON Payload, page 3-50

Invoking a REST Service Using HTTP Basic Authentication and XML Payload With REST Header

REST Service Invocation Scenario

Consider a PL/SQL API 'Profile Management APIs' (`FND_PROFILE`) as an example to explain the REST service invocation.

When a request is received to get the current value of a specific user profile option, a Java client is used to invoke the Get Profile REST service operation contained in the API. In this example, the request provides username and password information in the HTTP header, the user credentials are authenticated and authorized. After validation, the Get Profile REST service operation can be invoked for the authenticated user.

After the successful service invocation, the client will receive a REST response message with the profile value for the request. If the profile does not exist, null will return.

Prerequisites to Use a PL/SQL REST Service

Before invoking the PL/SQL REST service, ensure the following tasks are in place:

Setting Variables in RESTHeader for an HTTP Request

Applications context values can be passed in the 'RESTHeader' element before invoking a REST service that requires these values.

These context elements for PL/SQL interface type are *`Responsibility`*, *`RespApplication`*, *`SecurityGroup`*, *`NLSLanguage`*, and *`Org_Id`*.

The expected values for these elements are described in the following table:

Header Variables and Expected Values for PL/SQL Interface Type

Element Name	Expected Value
`Responsibility`	`responsibility_key` (such as "SYSTEM_ADMINISTRATOR")
`RespApplication`	Application Short Name (such as "SYSADMIN")
`SecurityGroup`	Security Group Key (such as "STANDARD")
`NLSLanguage`	NLS Language (such as "AMERICAN")
`Org_Id`	Org Id (such as "202")

Note: NLS Language and Org_Id are optional values to be passed.

- If the NLS Language element is specified, REST requests can be consumed in the language passed. All corresponding REST responses and error messages can also be returned in the same language. If no language is identified, then the default language of the user will be used.

- If a service execution is dependent on any particular organization, then you must pass the Org_Id element of that REST request.

Invoking a REST Service Using Java

Based on the REST service invocation scenario, the following tasks are included in this section:

1. Deploying a PL/SQL REST Web Service, page 3-44
2. Recording Resource Information from Deployed WADL, page 3-45
3. Creating a Project with a Java Class, page 3-46
4. Invoking a REST Service Using a Java Class, page 3-50

Deploying a PL/SQL REST Web Service

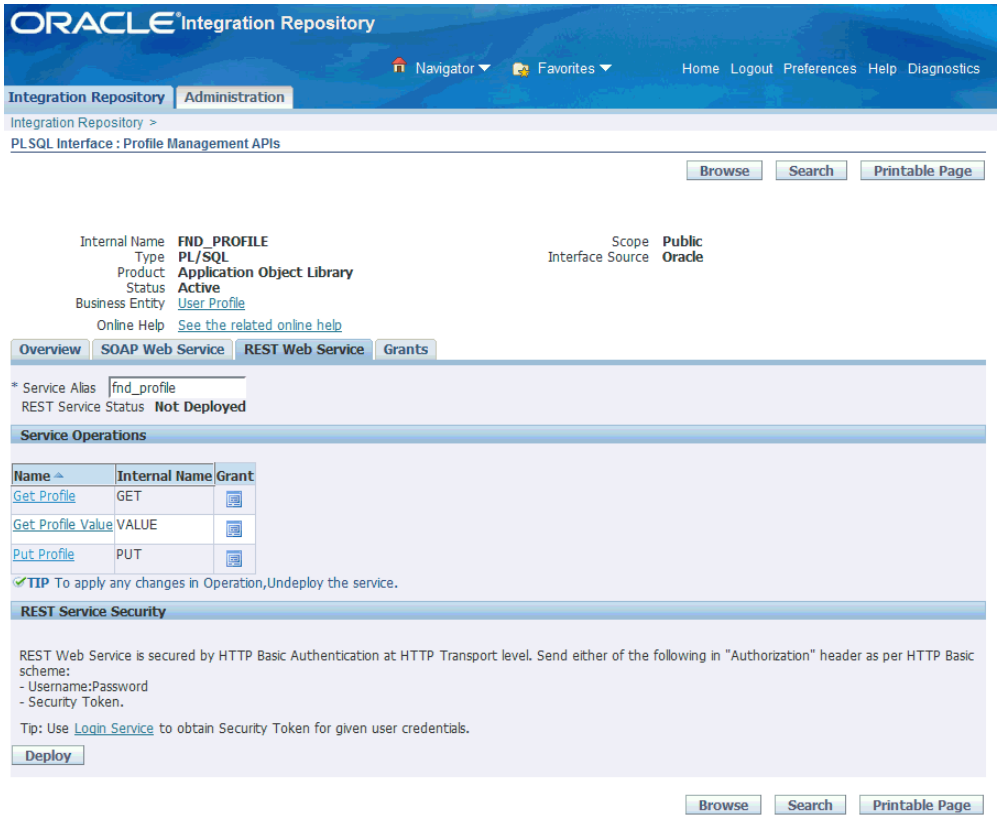
Use the following steps to deploy the 'Profile Management APIs' (`FND_PROFILE`):

1. Log in to Oracle E-Business Suite as a user who has the Integration Repository Administrator role. Select the Integrated SOA Gateway responsibility and then choose the Integration Repository link from the navigation menu.
2. In the Integration Repository tab, click **Search** to access the main Search page.
3. Enter 'FND_PROFILE' in the Internal Name field. Click **Go** to execute the search.

   Click the 'Profile Management APIs' interface name link to open the interface details page.
4. In the REST Web Service tab, enter "fnd_profile" in the Service Alias field.

   The alias will be displayed as the service endpoint in the WADL and schema.

Interface Details Page with REST Web Service Tab



5. Click **Deploy** to deploy the service to an Oracle E-Business Suite environment.

Once the REST service has been successfully deployed, 'Deployed' appears in the REST Service Status field along with the **View WADL** link. Click the **View WADL** link to view the deployed service WADL description.

For more information on deploying REST services, see Deploying REST Web Services, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

Recording Resource Information from Deployed WADL

To obtain service resource information from the deployed WADL for the `FND_PROFILE` service, an integration developer clicks the **View WADL** link in the REST Web Service tab.

The following WADL description appears:

```
<xml version="1.0" encoding="UTF-8" standalone="no" ?>
<application xmlns:tns="http://xmlns.oracle.
com/apps/fnd/soaprovider/plsql/rest/fnd_profile/" xmlns="http://wadl.
dev.java.net/2009/02" xmlns:tns1="http://xmlns.oracle.
com/apps/fnd/rest/fndprofilesvc/get/" name="FND_PROFILE"
targetNamespace="http://xmlns.oracle.
com/apps/fnd/soaprovider/plsql/rest/fnd_profile/">
<grammars>
    <include xmlns="http://www.w3.org/2001/XMLSchema" href="https:
//<hostname>:<port>/webservices/rest/FndProfileSvc/?XSD=GET.xsd" />
     <include xmlns="http://www.w3.org/2001/XMLSchema" href="https:
//<hostname>:<port>/webservices/rest/FndProfileSvc/?XSD=PUT.xsd" />
<include xmlns="http://www.w3.org/2001/XMLSchema" href="https:
//<hostname>:<port>/webservices/rest/FndProfileSvc/?XSD=VALUE.xsd" />
</grammars><resources base="http://<hostname>:
<port>/webservices/rest/fnd_profile/"><resource path="/get/">
   <method id="GET" name="POST">
    <request>
     <representation mediaType="application/xml" type="tns1:
InputParameters" />
     <representation mediaType="application/json" type="tns1:
InputParameters" />
    </request>
    <response>
     <representation mediaType="application/xml" type="tns1:
OutputParameters" />
     <representation mediaType="application/json" type="tns1:
OutputParameters" />
    </response>
   </method>
  </resource>
...
 </resources>
</application>
```

Copy or record the following information which will be used later when defining a Java client:

- `<resources base>="http://<hostname>:<port>/webservices/rest/fnd_profile/">`

  This information `http://<hostname>:<port>/webservices/rest/fnd_profile` will be used later in Java client program as the base URL.

- `<resource path>="/get/">`

  This information `/get/` will be used later to form the later part of the service URL.

Creating a Project with a Java Class

This section describes how to create a project with a Java class that will be used to invoke the `FND_PROFILE` REST service.

To create a project and a Java class:

1. Open Oracle JDeveloper.

2. From the main menu, choose **File > New**.

   In the New Gallery window, expand the General category and select 'Applications'.

In the Items list, select **Custom Application**.

Click **OK**. The "Create Custom Application - Name your application" page is displayed.

3. Enter an appropriate name for the application in the Application Name field. Click **Next**.

4. The "Create Custom Application - Name your project" page is displayed. Enter an appropriate name for the project in the Project Name field, for example 'ISGRESTClient'.

   In the Project Features tab, select '**Java**' from the Available list. Move the selected feature from the "Available" window to the "Selected" window using the right arrow button.

   Click **Next**.

5. Click **Finish** in the Configure Java Settings dialog box.

   The newly created project should be visible in the Projects workspace.

6. Select and right-click on the project name you just created in the Application Navigator and choose **New** from the drop-down selection menu.

7. In the New Gallery window, expand the General category and select 'Java'. In the Items list, select **Class**. Click **OK**.

8. In the Create Java Class dialog, change the default class name to 'RestInvocationBasicAuthWithHeader'. Accept all other defaults and click **OK**.

9. The new class opens automatically in the source editor, displaying the skeleton class definition.

   Replace the skeleton class definition with the following Java code:

```
package sample;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.net.HttpURLConnection;
import java.net.URL;
import com.sun.jersey.core.util.Base64;

public class RestInvocationBasicAuthWithHeader {
 // xml payload with REST header for invoking the service
 private static final String xmlRequest4 = "<ns:GET_Input xmlns:
ns=\"http://xmlns.oracle.
com/apps/fnd/soaprovider/plsql/rest/fnd_profile/get/\""
   + "     xmlns:ns1=\"http://xmlns.oracle.
com/apps/fnd/soaprovider/plsql/rest/fnd_profile/header\">"
   + "    <ns1:RESTHeader>"
   + "   <ns1:Responsibility>SYSTEM_ADMINISTRATOR</ns1:
Responsibility>"
   + "   <ns1:RespApplication>SYSADMIN</ns1:RespApplication>"
   + "   <ns1:SecurityGroup>STANDARD</ns1:SecurityGroup>"
   + "   <ns1:NLSLanguage>AMERICAN</ns1:NLSLanguage>"
   + "   <ns1:Org_Id>202</ns1:Org_Id>"
   + "    </ns1:RESTHeader>"
   + "    <ns:InputParameters>"
   + "    <ns:NAME>APPS_SERVLET_AGENT</ns:NAME>"
   + "    </ns:InputParameters> + </ns:GET_Input>";

 /**
  * This method invokes a REST service using basic Authentication
and xml payload with REST headers.
  */
 public static void postXml_BasicAuth(String svcUrlStr, String
username,String passwd) throws IOException {

  URL url = new URL(svcUrlStr);
  // Obtaining connection to invoke the service
  HttpURLConnection conn = (HttpURLConnection) url.openConnection();
  // Setting Http header values
  conn.setRequestMethod("POST");
  conn.setRequestProperty("Content-Type", "application/xml");
  String auth = username + ":" + passwd;
  byte[] bytes = Base64.encode(auth);
  String authStr = new String(bytes);
  conn.setRequestProperty("Authorization", "Basic " + authStr);
  conn.setRequestProperty("Accept", "application/xml");
  conn.setRequestProperty("Content-Language", "en-US");
  conn.setUseCaches(false);
  conn.setDoInput(true);
  conn.setDoOutput(true);
  // Send request
  OutputStreamWriter wr = new OutputStreamWriter(conn.
getOutputStream());
  wr.write(xmlRequest4.toCharArray());
  wr.flush();
  wr.close();
  conn.connect();
  System.out.println("Response code - " + conn.getResponseCode());
  // Get Response
  String response = null;
  try {
   response = readHttpResponse(conn);
  } finally {
```

```
if (conn != null)
    conn.disconnect();
  }
  // Show response
  System.out.println("Response is : \n" + response);
 }

 /**
  * This method reads response from server and returns it in a
string representation.
  */
 private static String readHttpResponse(HttpURLConnection conn) {

  InputStream is = null;
  BufferedReader rd = null;
  StringBuffer response = new StringBuffer();
  try {

   if (conn.getResponseCode() >= 400) {
    is = conn.getErrorStream();
   } else {
    is = conn.getInputStream();
   }
   rd = new BufferedReader(new InputStreamReader(is));
   String line;
   while ((line = rd.readLine()) != null) {
    response.append(line);
    response.append('\n');
   }
  } catch (IOException ioe) {
   response.append(ioe.getMessage());
  } finally {
   if (is != null) {
    try {
     is.close();
    } catch (Exception e) {
    }
   }
   if (rd != null) {
    try {
     rd.close();
    } catch (Exception e) {
    }
   }
  }
  return (response.toString());
 }

 public static void main(String a[]) {
  String baseUrl = "http://<server hostname>:<port>/webservices/rest
";
  String svcUrlStr1 = baseUrl + "/FndProfileSvc/get/";
  // invoke Rest service using basic authentication method
  try {
   postXml_BasicAuth(svcUrlStr1, "SYSADMIN", "sysadmin");
  } catch (IOException e) {
   e.printStackTrace();
  }
 }
}
```

Please note that resource information recorded earlier from the deployed WADL is now placed in the `baseUrl` and `svcUrlStr1` elements.

Note: Use **https** (instead of http) in the `baseUrl` if your Oracle E-Business Suite instance is running on the SSL-enabled environment. Additionally, you need to import the SSL certificate into your client JVM's keystore.

10. Replace `<server hostname>:<port>` with the actual values in the code.

11. Save your work by selecting **File > Save All**.

Invoking a REST Service Using a Java Class

After creating a project with a Java class `RestInvocationBasicAuthWithHeader.java`, you need to compile and execute the process to invoke the REST service.

Use the following steps to compile and run the Java class:

1. In the Application Navigator, right-click on the `RestInvocationWithLogin.java` Java class you just created at the design time. Select **Make** from the menu.

2. Right-click on the `RestInvocationBasicAuthWithHeader.java` Java class. Select **Run** from the menu.

Monitor this process and check for successful compilation in the Log window. Verify that the execution is successful in the Log window.

Viewing Output Message

When the REST service is successfully invoked, the following output appears in the Log window:

```
Response code - 200
Response is :
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<OutputParameters xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.oracle.com/apps/fnd/rest/fnd_profile/get/">
<VAL>http://<server hostname>:<port>/OA_HTML</VAL>
</OutputParameters>
```

The value of the profile option 'APPS_SERVLET_AGENT' is obtained by the service and displayed in the `<VAL></VAL>` tag.

Notice that service alias information **fnd_profile** entered earlier during service deployment appears as part of the service endpoint.

Invoking a REST Service Using Token Based Authentication and JSON Payload

REST Service Invocation Scenario

A PL/SQL API User (`FND_USER_PKG`) is used in this example to explain the REST service invocation.

When a consequent HTTP request is received from the same user to request for testing

user names in Oracle E-Business Suite, the Test User Name (TESTUSERNAME) REST service operation contained in the API is invoked to test the users against the FND_USER table.

Since password is not provided in this request, token based security method is used to authenticate the user credentials. The security Login service is launched to create an Oracle E-Business Suite user session and returns the session ID as cookie in place of password for user authentication. After validation, the Test User Name (TESTUSERNAME) REST service operation can be invoked.

In this example, user name information to be tested is passed in a JSON-based payload for REST service invocation. When the service has been successfully executed, the TESTUSERNAME operation returns a positive number if the user name passed in the payload exists in Oracle E-Business Suite. If the user name does not exist, then number 0 is returned instead.

Prerequisites to Use a PL/SQL REST Service

Before performing the design-time tasks, ensure the following tasks are in place:

Obtaining Needed Libraries

To successfully invoke the REST service with payload in JSON format, you need to obtain the following libraries available at <$COMMON_TOP>/java/lib directory:

- jersey-bundle_1.0.0.0_1-1-5-1.jar
- jackson-core-asl_1.0.0.0_1-1-1.jar
- jackson-mapper-asl_1.0.0.0_1-1-1.jar

These library files will be added to the project later at the design-time during the project creation.

Setting Variables in RESTHeader for an HTTP Request

In REST services, applications context values can be passed in the 'RESTHeader' element before invoking a REST service.

These RESTHeader elements for PL/SQL interface type are *Responsibility*, *RespApplication*, *SecurityGroup*, *NLSLanguage*, and *Org_Id*.

For more information about the RESTHeader elements for PL/SQL interface type, see Invoking a REST Service Using HTTP Basic Authentication and XML Payload With REST Header, page 3-42.

Invoking a REST Service Using Java

Based on the REST service invocation scenario, the following tasks are included in this section:

1. Deploying a PL/SQL REST Web Service, page 3-52
2. Recording the Deployed WADL URL, page 3-53

3. Creating a Project with a Java Class, page 3-55

4. Invoking a REST Service Using a Java Class, page 3-62

Deploying a PL/SQL REST Web Service

Use the following steps to deploy the User API (`FND_USER_PKG`):

1. Log in to Oracle E-Business Suite as a user who has the Integration Repository Administrator role. Select the Integrated SOA Gateway responsibility and then choose the Integration Repository link from the navigation menu.

2. In the Integration Repository tab, click **Search** to access the main Search page.

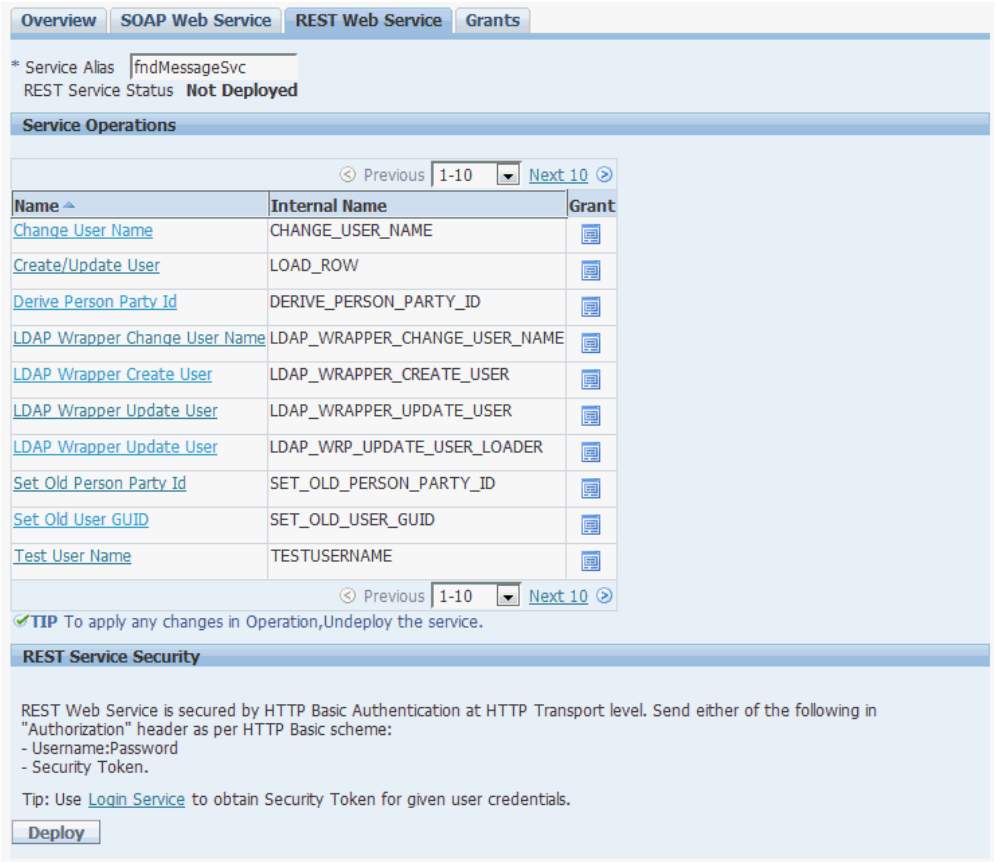
3. Enter 'FND_USER_PKG' in the Internal Name field. Click **Go** to execute the search.

   Click the 'User' interface name link to open the interface details page.

4. In the REST Web Service tab, enter "fndMessageSvc" in the Service Alias field.

   The alias will be displayed as the service endpoint in the WADL and schema which is TESTUSERNAME in this example.

REST Web Service Tab for Entering Service Alias



5. Click **Deploy** to deploy the service to an Oracle E-Business Suite environment.

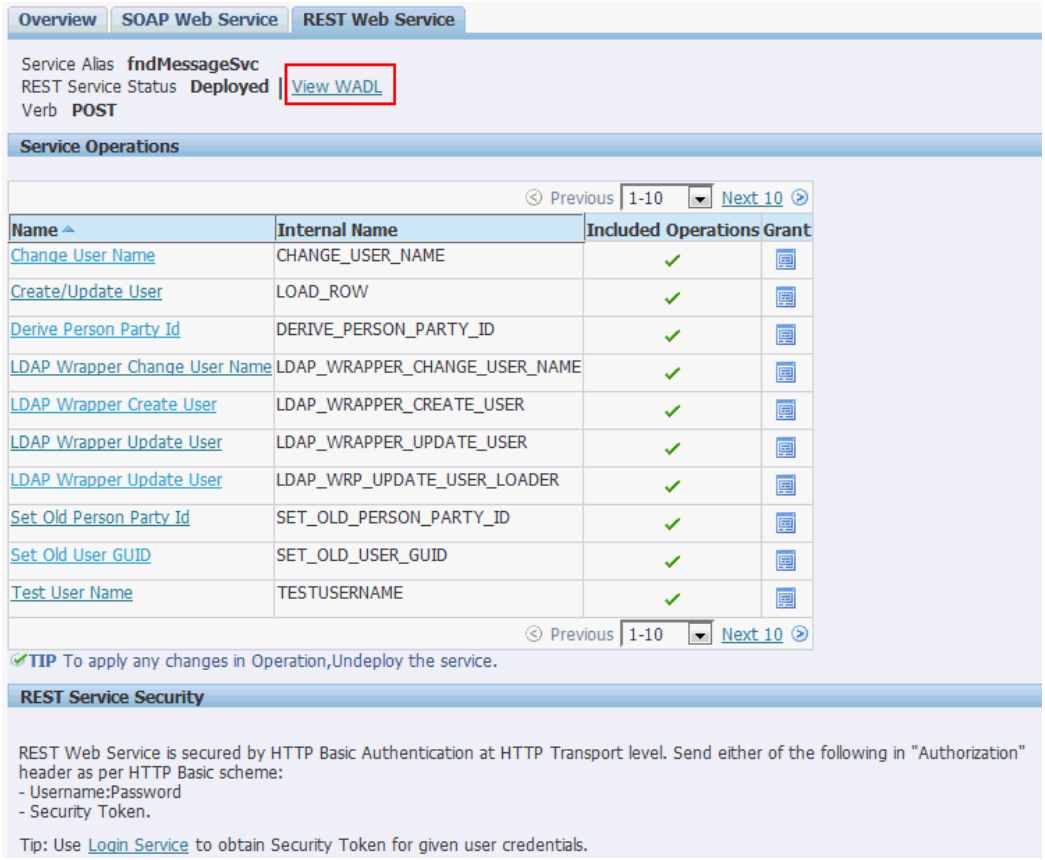
Once the REST service has been successfully deployed, 'Deployed' appears in the REST Service Status field along with the **View WADL** link allowing you to view the WADL description.

For more information on deploying REST services, see Deploying REST Web Services, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

Recording the Deployed WADL URL

To obtain service resource information from the deployed WADL for the `FND_USER_PKG` service, an integration developer clicks the **View WADL** link in the REST Web Service tab.

REST Web Service Tab with Deployed "View WADL" Link Highlighted



The following WADL description appears:

```
<xml version="1.0" encoding="UTF-8" standalone="no" ?>
<application xmlns:tns="http://xmlns.oracle.
com/apps/fnd/soaprovider/plsql/rest/fnd_user_pkg/" xmlns="http://wadl.
dev.java.net/2009/02" xmlns:tns1="http://xmlns.oracle.
com/apps/fnd/rest/fndMessageSvc/testusername/" name="FND_USER_PKG"
targetNamespace="http://xmlns.oracle.
com/apps/fnd/soaprovider/plsql/rest/fnd_user_pkg/">
<grammars>
    <include xmlns="http://www.w3.org/2001/XMLSchema" href="https:
//<hostname>:<port>/webservices/rest/fndMessageSvc/?
XSD=TESTUSERNAME_SYNCH_TYPEDEF.xsd" />
...
</grammars><resources base="http://<hostname>:
<port>/webservices/rest/fndMessageSvc/"><resource path="/testusername/">
   <method id="GET" name="POST">
    <request>
     <representation mediaType="application/xml" type="tns1:
InputParameters" />
     <representation mediaType="application/json" type="tns1:
InputParameters" />
    </request>
    <response>
     <representation mediaType="application/xml" type="tns1:
OutputParameters" />
     <representation mediaType="application/json" type="tns1:
OutputParameters" />
    </response>
   </method>
  </resource>
...
 </resources>
</application>
```

Copy or record the following information which will be used later when defining a Java client:

- `<resources base>="http://<hostname>:<port>/webservices/rest/fndMessageSvc/">`

  This information `http://<hostname>:<port>/webservices/rest/fndMessageSvc` will be used later in Java client program as the base URL.

- `<resource path>="/testusername/">`

  This information will be used later to form the later part of the service URL.

Creating a Project with a Java Class

This section describes how to create a project with a Java class (`RestInvocationWithLogin.java`) and JSON payload that will be used to invoke the `FND_USER_PKG` REST service.

To create a project with a Java class:

1. In Oracle JDeveloper, choose **File > New** from the main menu.

   In the New Gallery window, expand the General category and select 'Applications'. In the Items list, select **Custom Application**.

Click **OK**. The "Create Custom Application - Name your application" page is displayed.

2. Enter an appropriate name for the application in the Application Name field. Click **Next**.

3. The "Create Custom Application - Name your project" page is displayed. Enter an appropriate name for the project in the Project Name field, for example 'ISGRESTClient3'.

   In the Project Features tab, select '**Java**' from the Available list. Move the selected feature from the "Available" window to the "Selected" window using the right arrow button.

   Click **Next**

4. Click **Finish** in the Configure Java Settings dialog box.

5. In the Application Navigator and right-click on the project you just created, and choose **New** from the drop-down menu.

6. In the New Gallery window, expand the General category and select 'Java'. In the Items list, select **Class**. Click **OK**.

7. In the Create Java Class dialog, change the default class name to 'RestInvocationWithLogin'. Accept all other defaults and click **OK**.

8. The new class opens automatically in the source editor, displaying the skeleton class definition.

   Replace the skeleton class definition with the following Java code:

```
package sample;

import java.io.BufferedReader;
import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.net.HttpURLConnection;
import java.net.URL;
import org.codehaus.jackson.JsonNode;
import org.codehaus.jackson.JsonParseException;
import org.codehaus.jackson.JsonParser;
import org.codehaus.jackson.map.ObjectMapper;
import com.sun.jersey.core.util.Base64;

public class RestInvocationWithLogin {

 private static final String jsonRequest1 = "{\"
TESTUSERNAME_Input\":{ "
  + "   \"@xmlns\":\"http://xmlns.oracle.com/apps/fnd/rest/
fndMessageSvc/testusername/\","
  + "   \"RESTHeader\":{ "
  + "     \"@xmlns\":\"http://xmlns.oracle.
com/apps/fnd/rest/fndMessageSvc/header\","
  + "     \"Responsibility\":\"SYSTEM_ADMINISTRATOR\","
  + "     \"RespApplication\":\"SYSADMIN\","
  + "     \"SecurityGroup\":\"STANDARD\","
  + "     \"NLSLanguage\":\"AMERICAN\","
  + "     \"Org_Id\":\"202\" "
  + "   }, " + "   \"InputParameters\":{ "
  + "     \"X_USER_NAME\":\"operations\" " + "   }" + "}}";


 /**
  * This Method invokes the a rest service using the accessTokenName
and accessToken values returned by AOL login service
  */
 public static void postJSON_AolToken(String svcUrlStr, String
tokenName,String tokenValue) throws IOException {

  URL url = new URL(svcUrlStr);
  //Obtaining connection to invoke the service
  HttpURLConnection conn = (HttpURLConnection) url.openConnection();
  //Setting Http header values
  conn.setRequestMethod("POST");
  conn.setRequestProperty("Content-Type", "application/json");
  //Adding the accessTokenName and accessToken as Cookies
  conn.addRequestProperty("Cookie", tokenName + "=" + tokenValue);
  conn.setRequestProperty("Accept", "application/json");
  conn.setRequestProperty("Content-Language", "en-US");
  conn.setUseCaches(false);
  conn.setDoInput(true);
  conn.setDoOutput(true);
  //Send request
  OutputStreamWriter wr = new OutputStreamWriter(conn.
getOutputStream());
  wr.write(jsonRequest1.toCharArray());
  wr.flush();
  wr.close();
  conn.connect();
  System.out.println("Response code - " + conn.getResponseCode());
  //Get Response
  String response = null;
  try {
```

```
response = readHttpResponse(conn);
  } finally {
   if (conn != null)
    conn.disconnect();
  }
  //Show Response
  System.out.println("Response is : \n" + response);
 }

 /**
  * This method invokes the AOL login service.It authenticates login
credentials and  returns accessTokenName and accessToken values
after successful validation of login credentials.
  */
 private static String[] getAolToken(String baseUrl, String
username,String passwd) throws Exception {

  String rfUrl = baseUrl + "/login";
  URL url = new URL(rfUrl);
  //Obtaining connection to invoke login service.
  HttpURLConnection conn = (HttpURLConnection) url.openConnection();
  String auth = username + ":" + passwd;
  byte[] bytes = Base64.encode(auth);
  String authStr = new String(bytes);
  //Setting Http request method
  conn.setRequestMethod("GET");
  //Setting the Http header values
  conn.setRequestProperty("Authorization", "Basic " + authStr);
  conn.setRequestProperty("Content-type", "application/json");
  conn.setRequestProperty("Accept", "application/json");
  conn.setUseCaches(false);
  conn.setDoInput(true);
  conn.setDoOutput(true);
  conn.connect();
  String response = null;
  //Get Response
  try {
   response = readHttpResponse(conn);
  } finally {
   if (conn != null)
    conn.disconnect();
  }
  //Parsing Response to obtain
  JsonParser jp = null;
  JsonNode root = null;
  ObjectMapper mapper = new ObjectMapper();
  try {
   jp = mapper.getJsonFactory().createJsonParser(new
ByteArrayInputStream(response.getBytes()));
   jp.disableFeature(org.codehaus.jackson.JsonParser.Feature.
AUTO_CLOSE_SOURCE);
   root = jp.readValueAsTree();
  } catch (JsonParseException jpe) {
   jpe.printStackTrace();
  } catch (IOException ioe) {
   ioe.printStackTrace();
  }
  JsonNode dataNode = root.get("data");
  JsonNode accessTokenNode = dataNode.get("accessToken");
  String accessToken = accessTokenNode.getTextValue();
  JsonNode accessTokenNameNode = dataNode.get("accessTokenName");
  String accessTokenName = accessTokenNameNode.getTextValue();
  return (new String[] { accessTokenName, accessToken });
 }
```

```
/**
  * This method reads response sent by the server and returns it in
a string representation.
  */
 private static String readHttpResponse(HttpURLConnection conn) {
  InputStream is = null;
  BufferedReader rd = null;
  StringBuffer response = new StringBuffer();
  try {
   if (conn.getResponseCode() >= 400) {
    is = conn.getErrorStream();
   } else {
    is = conn.getInputStream();
   }
   rd = new BufferedReader(new InputStreamReader(is));
   String line;
   while ((line = rd.readLine()) != null) {
    response.append(line);
    response.append('\n');
   }
  } catch (IOException ioe) {
   response.append(ioe.getMessage());
  } finally {
   if (is != null) {
    try {
     is.close();
    } catch (Exception e) {
    }
   }
   if (rd != null) {
    try {
     rd.close();
    } catch (Exception e) {
    }
   }
  }
  return (response.toString());
 }

 public static void main(String[] args) throws Exception {
  String baseUrl = "http://<server hostname>:<port>/webservices/rest
";
  String svcUrlStr1 = baseUrl + "/fndMessageSvc/testusername/";
  //Get Access Token by invoking AOL Login Service
  String[] token = getAolToken(baseUrl, "SYSADMIN", "sysadmin");
  System.out.println("AOL Token : Name - " + token[0] + ", Value -
"+ token[1]);
  //Invoke REST service using the Access Token
  postJSON_AolToken(svcUrlStr1, token[0], token[1]);
 }
}
```

Please note that resource information recorded earlier from the deployed WADL is now placed in the `baseUrl` and `svcUrlStr1` elements.

> **Note:** Use **https** (instead of http) in the `baseUrl` if your Oracle E-Business Suite instance is running on the SSL-enabled environment. Additionally, you need to import the SSL certificate into your client JVM's keystore.

9. Replace `<server hostname>:<port>` with the actual values in the code.

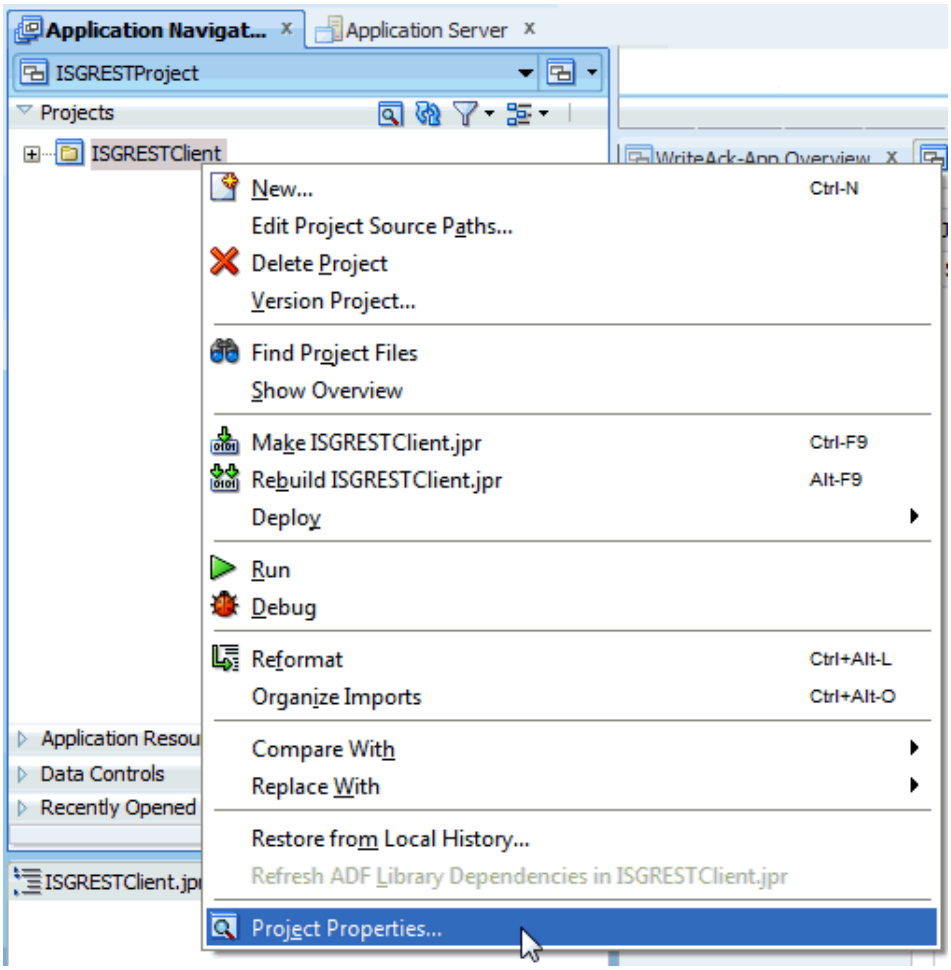
10. **Add required libraries to process JSON payload:**

    Use the following steps to add the required library files to the project properties.

    1. Select and right-click on the project name you just created earlier to open a selection menu.

    2. Select **Project Properties** from the menu.

Drop-Down Menu to Select Project Properties



The Default Properties dialog box opens.

3. Select **Libraries and Classpath**, and click **Add Library**. The Add Library dialog box opens.

4. In the Add Library dialog box, select the Project folder and then click **New**.

   The Create Library dialog box opens.

5. In the Library Name field, enter 'jackson-core-asl_1.0.0.0_1-1-1.jar'.

   Click **Add Entry**. The Select Path Entry dialog box appears.

6. In the Select Path Entry dialog box, locate and select the 'jackson-core-asl_1.0.0.0_1-1-1.jar' file that you have downloaded. This adds it to the Classpath.

   Click **OK**. The 'ackson-core-asl_1.0.0.0_1-1-1.jar' is now added to the Project

folder.

7. Repeat steps 4, 5, and 6 to add the following two jar files to the Project folder:
   - jersey-bundle_1.0.0.0_1-1-5-1.jar
   - jackson-mapper-asl_1.0.0.0_1-1-1.jar

   These three jar files should now appear in the Project folder. Click **OK**.

8. The Project Properties dialog box appears. Click **OK**. This project is now set up with the required libraries.

11. Save your work by selecting **File > Save All**.

Invoking REST Service Using a Java Client

After creating a project with a Java class `RestInvocationWithLogin.java`, you need to compile and execute the process to invoke the `FND_USER_PKG` REST service.

Use the following steps to compile and run the Java class:

1. In the Application Navigator, right-click on the `RestInvocationWithLogin.java` Java class you just created at the design time. Select **Make** from the menu.

2. Right-click on the `RestInvocationWithLogin.java` Java class and select **Run** from the menu.

Monitor this process and check for successful compilation in the Log window. Verify that the execution is successful in the Log window.

Viewing Output Message

When the `FND_USER_PKG` REST service is successfully invoked, the following output appears:

- The response from the Login service should be like:

```
AOL Token : Name - isgdemo, Value - xxxxxxxxxxxxxxxxxxxxxxxxxx
```

- The response from the service invocation should be like:

```
Response code - 200
Response is :
{
  "OutputParameters" : {
    "@xmlns:xsi" : "http://www.w3.org/2001/XMLSchema-instance",
    "@xmlns" : "http://xmlns.oracle.com/apps/fnd/rest/fndMessageSvc
/testusername/",
    "TESTUSERNAME" : "2"
  }
}
```

In this example, a positive number '2' is returned indicating that the user name

passed in the payload does exist in Oracle E-Business Suite.

Notice that service alias information **fndMessageSvc** entered earlier during service deployment appears as part of the service endpoint.

- The response from the Logout service should be like:

```
Response is :
{
  "data" : {
    "accessToken" : "-1",
    "accessTokenName" : "isgdemo",
    "ebsVersion" : null
  }
}
```

4

Using Java Bean Services as REST Services

Overview

Java APIs are business interfaces based on Java classes. Some specialized Java APIs whose methods must use parameters of either serializable Java Beans or simple data types such as `String`, `Int`, and so forth can be categorized as Java Bean Services.

Java Bean Services are a subtype of Java interface. You can locate them through a search by "Interface Subtype" category and "Java Bean Services" category value. Oracle E-Business Suite Integrated SOA Gateway allows these specialized Java classes to be exposed as REST services.

Similar to PL/SQL REST services, once Java Bean Services have been successfully deployed as REST services, an integration developer can invoke the deployed REST services from client program using languages like Java, PHP, Javascript, Python, and so on.

To better understand how to use the deployed service to fetch and use application data, detailed design-time and runtime tasks are included in this chapter. For the example described in the following sections, Oracle JDeveloper 10g (10.1.3.3.0) is used to create a project with a Java class and invoke the service.

The following REST service invocation examples are included in this section:

Invoking a REST Service Using HTTP GET Method

REST Service Invocation Scenario

Consider a Java Bean service 'REST Service Locator' (`oracle.apps.fnd.rep.ws.service.EbsRestLocator`) as an example to explain the REST service invocation. REST Service Locator is a sample Java API that consists of methods to retrieve details

about deployed Oracle E-Business Suite REST services.

A Java client is used to make HTTP GET request to the `getRestInterface` service operation. The `getRestInterface` service operation returns the details of a REST service identified by its internal name.

In this example, HTTP Basic Authentication scheme is used to provide username and password information in the HTTP request header. The user credentials are authenticated and authorized by ISG's REST Service Provider. After validation, the `getRestInterface` service operation executes the request for the authenticated user.

After the successful service invocation, the client will receive a REST response message with the details of the REST service whose internal name has been passed in HTTP URL at run time during service invocation.

Invoking a REST Service Using Java

Based on the REST service invocation scenario, the following tasks are included in this section:

1. Deploying a REST Service, page 4-2
2. Creating a Security Grant, page 4-4
3. Recording Resource Information from Deployed WADL, page 4-6
4. Creating a Project with a Java Class, page 4-9
5. Invoking a REST Service Using a Java Class, page 4-13

Deploying a REST Service

Use the following steps to deploy the Java Bean Service called REST Service Locator:

1. Log in to Oracle E-Business Suite as a user who has the Integration Repository Administrator role.

   Select the Integrated SOA Gateway responsibility and the Integration Repository link from the navigation menu.

2. In the Integration Repository tab, click **Search** to access the main Search page.

3. Click **Show More Search Options** to display more search fields.

   Enter the following key search values as the search criteria:

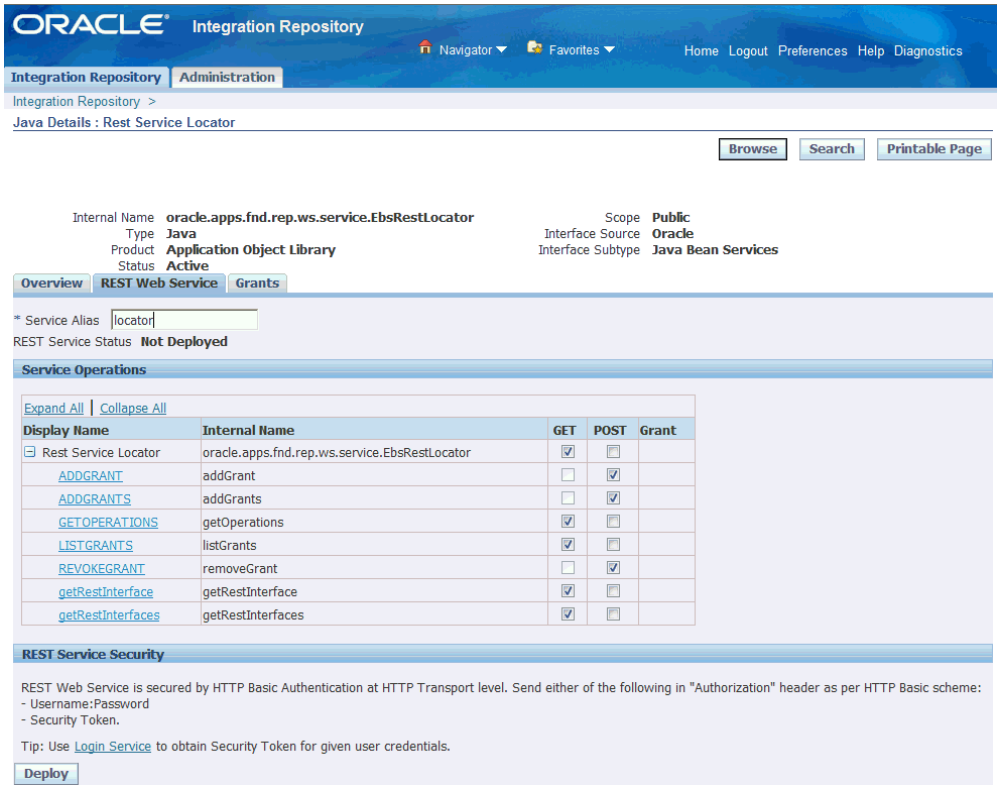
   - Category: Interface Subtype
   - Category Value: Java Bean Services

4. Click **Go** to execute the search.

Click the REST Service Locator interface name link to open the interface details page.

5. In the REST Web Service tab, enter the following information:

Interface Details Page with REST Web Service Tab to Deploy a Java REST Service



- Service Alias: locator

  The alias will be displayed as the service endpoint in the WADL and schema for the selected method or operation.

- Select Desired Service Operations

  In the Service Operations region, HTTP method check boxes are preselected.

  Note that if a Java method is annotated with a specific HTTP method, then the corresponding HTTP method check box is preselected for that method. The administrator can change the HTTP method check box selection before deploying the service.

  For more Java Bean Services annotation guidelines, see Annotations for Java Bean Services, page A-6.

  In this example, the 'getRestInterface' service operation has been annotated with the GET HTTP method; therefore, the GET check box is automatically

selected.

6. Click **Deploy** to deploy the service to an Oracle E-Business Suite environment.

Once the REST service has been successfully deployed, 'Deployed' appears in the REST Service Status field along with the **View WADL** link. Click the **View WADL** link to view the deployed service WADL description.

For more information on deploying REST services, see Deploying REST Web Services, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

Creating a Security Grant

After deploying the REST Service Locator as a REST service, the integration administrator can create a security grant to authorize the service or method access privileges to a specific user, a user group, or all users.

Use the following steps to create a security grant:

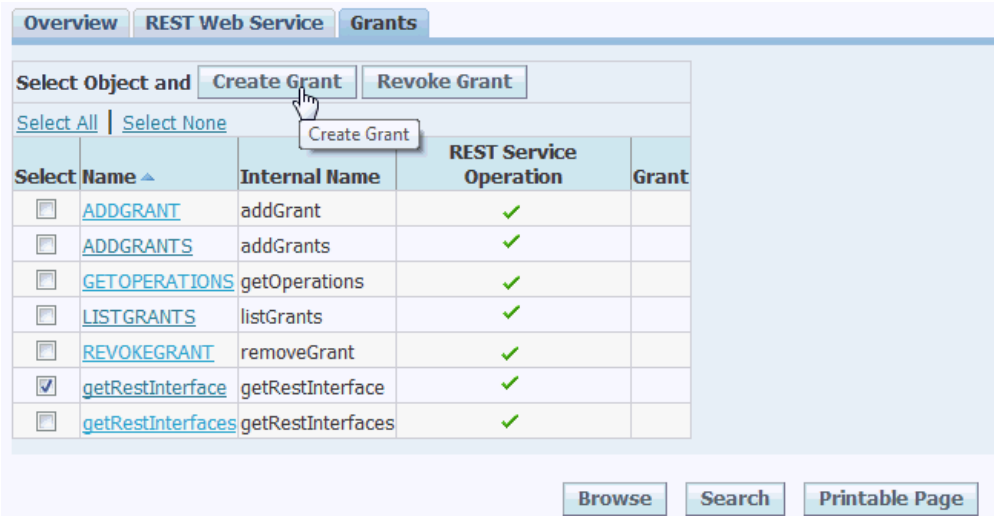
1. Log in to Oracle E-Business Suite as a user who has the Integration Repository Administrator role. Select the Integrated SOA Gateway responsibility and the Integration Repository link from the navigation menu.

2. Perform a search to locate the REST Service Locator service the administrator just deployed earlier.

3. In the interface details page of the selected custom Java Bean Services, click the Grants tab.

4. Select the `getRestInterface` method check box and then click **Create Grant**.

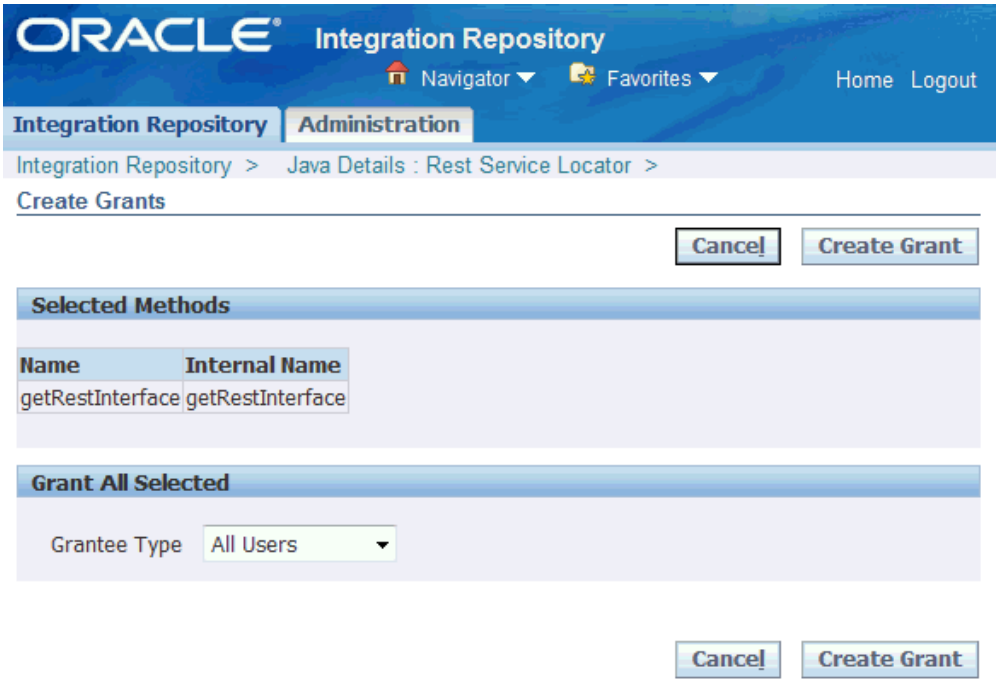
Interface Details Page with Grants Tab



5. In the Create Grants page, select "All Users" as the Grantee Type.

   Note: In this example, the `getRestInterface` service operation is granted to all users. In actual implementation, you should define strict security rules. Create grant to a user or more appropriately to a group of users defined by roles and responsibilities.

Create Grants Page

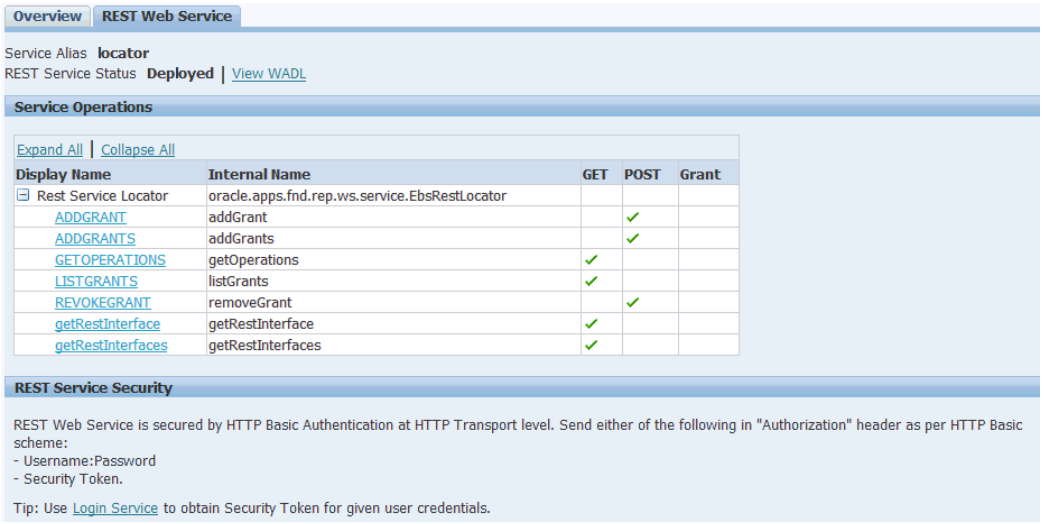


Click **Create Grant**. This grants the selected method access privilege to all Oracle E-Business Suite users.

Recording Resource Information from Deployed WADL

To obtain service resource information from the deployed WADL for the REST Service Locator service, an integration developer clicks the **View WADL** link in the REST Web Service tab.

REST Web Service Tab with Deployed "View WADL" Link



The following WADL description appears:

```
<xml version="1.0" encoding="UTF-8">
<application name="EbsRestLocator" targetNamespace="http://xmlns.oracle.
com/apps/fnd/soaprovider/pojo/ebsrestlocator/"
xmlns:tns="http://xmlns.oracle.
com/apps/fnd/soaprovider/pojo/ebsrestlocator/"
xmlns="http://wadl.dev.java.net/2009/02" xmlns:xsd="http://www.w3.
org/2001/XMLSchema"
xmlns:tns1="http://xmlns.oracle.com/apps/fnd/rest/locatorsvc/addgrant/"
xmlns:tns2="http://xmlns.oracle.com/apps/fnd/rest/locatorsvc/addgrants/"

xmlns:tns3="http://xmlns.oracle.
com/apps/fnd/rest/locatorsvc/getoperations/"
xmlns:tns4="http://xmlns.oracle.
com/apps/fnd/rest/locatorsvc/getrestinterface/"
xmlns:tns5="http://xmlns.oracle.
com/apps/fnd/rest/locatorsvc/getrestinterfaces/"
xmlns:tns6="http://xmlns.oracle.
com/apps/fnd/rest/locatorsvc/removegrant/">
<grammars>
    <include href="http://<hostname>:
<port>/webservices/rest/servicelocator/?XSD=addgrant.xsd" xmlns="http:
//www.w3.org/2001/XMLSchema" />
   <include href="http://<hostname>:
<port>/webservices/rest/servicelocator/?XSD=addgrants.xsd" xmlns="http:
//www.w3.org/2001/XMLSchema" />
   <include href="http://<hostname>:
<port>/webservices/rest/servicelocator/?XSD=getoperations.xsd" xmlns="
http://www.w3.org/2001/XMLSchema" />
   <include href="http://<hostname>:
<port>/webservices/rest/servicelocator/?XSD=getrestinterface.xsd"
xmlns="http://www.w3.org/2001/XMLSchema" />
   <include href="http://<hostname>:
<port>/webservices/rest/servicelocator/?XSD=getrestinterfaces.xsd"
xmlns="http://www.w3.org/2001/XMLSchema" />
   <include href="http://<hostname>:
<port>/webservices/rest/servicelocator/?XSD=removegrant.xsd" xmlns="
http://www.w3.org/2001/XMLSchema" />
</grammars><resources base="http://<hostname>:
<port>/webservices/rest/locator/">
  <resource path="addGrant/">
/
   ...
  </resource path>
...
  <resource path="/getRestInterface/{irepClassName}/">
    <param name="irepClassName" style="template" required="true" type="
xsd:string" />
  <method id="getRestInterface" name="GET">
    <request>
     <param name="ctx_responsibility" type="xsd:string" style="query"
required="false" />
              <param name="ctx_respapplication" type="xsd:string"
style="query" required="false" />
     <param name="ctx_securitygroup" type="xsd:string" style="query"
required="false" />
     <param name="ctx_nlslanguage" type="xsd:string" style="query"
required="false" />
     <param name="ctx_orgid" type="xsd:int" style="query" required="
false" />
    </request>
      <response>
     <representation mediaType="application/xml" type="tns4:
getRestInterface_Output" />
     <representation mediaType="application/json" type="tns4:
getRestInterface_Output" />
    </response>
```

```
</method>
  </resource>
<resource path="removeGrant/">
/
   ...
</resource path>
</application>
```

Copy or record the following information which will be used later when defining a Java client:

- `<resources base>="http://<hostname>:<port>/webservices/rest/locator/">`

  This information `http://<hostname>:<port>/webservices/rest/locator/` will be used later in Java client program as baseUrl.

- `<resource path>="/getRestInterface/{irepClassName}/">`

  This information `/getRestInterface/{irepClassName}/` will be used later to form the later part of the service URL. The input `{irepClassName}` is a path variable, and it will be replaced with the internal name of an interface.

Creating a Project with a Java Class

This section describes how to create a project with a Java class that will be used to invoke the REST Service Locator service.

To create a project and a Java class:

1. Open Oracle JDeveloper.

2. From the main menu, choose **File > New**.

   In the New Gallery window, expand the General category and select 'Applications'. In the Items list, select **Custom Application**.

   Click **OK**. The "Create Custom Application - Name your application" page is displayed.

3. Enter an appropriate name for the application in the Application Name field, for example ISGJavaRESTProject. Click **Next**.

4. The "Create Custom Application - Name your project" page is displayed. Enter an appropriate name for the project in the Project Name field, for example ISGJavaRESTClient.

   In the Project Features tab, select '**Java**' from the Available list. Move the selected feature from the "Available" window to the "Selected" window using the right arrow button.

   Click **Next**.

5. Click **Finish** in the Configure Java Settings dialog box.

   The newly created project should be visible in the Projects workspace.

6. Select and right-click on the project name you just created in the Application Navigator and choose **New** from the drop-down selection menu.

7. In the New Gallery window, expand the General category and select 'Java'. In the Items list, select **Class**. Click **OK**.

8. In the Create Java Class dialog, change the default class name to 'RestInvocationGETMethod'. Accept all other defaults and click **OK**.

9. The new class opens automatically in the source editor, displaying the skeleton class definition.

   Replace the skeleton class definition with the following Java code:

```
package isgrestget;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import com.sun.jersey.core.util.Base64;

public class RestInvocationGETMethod {

 /**
  * This Method invokes the a rest service using HTTP GET Method
with path parameter in URL
  */
 public static void invokeREST(String svcUrlStr,String username,
String passwd,String pathParam) throws IOException {
                String getURL = svcUrlStr + "/" + pathParam;
  URL url = new URL(getURL);
  //Obtaining connection to invoke the service
  HttpURLConnection conn = (HttpURLConnection) url.openConnection();
  String auth = username + ":" + passwd;
  byte[] bytes = Base64.encode(auth);
  String authStr = new String(bytes);

  //Setting Http header values
  conn.setRequestMethod("GET");
  conn.setRequestProperty("Authorization", "Basic " + authStr);
  conn.setRequestProperty("Accept", "application/json");
  conn.setRequestProperty("Content-Language", "en-US");
  conn.setUseCaches(false);
  conn.setDoInput(true);
  conn.setDoOutput(true);
  conn.connect();
         System.out.println("\n 'GET' request sent to URL : " +
url);
  System.out.println("\n Response code - " + conn.
getResponseCode());
  //Get Response
  String response = null;
  try {
   response = readHttpResponse(conn);
  } finally {
   if (conn != null)
    conn.disconnect();
  }
  //Show Response
  System.out.println("Response is : \n" + response);
 }

 /**
  * This method reads response sent by the server and returns it in
a string representation.
  */
 private static String readHttpResponse(HttpURLConnection conn) {
  InputStream is = null;
  BufferedReader rd = null;
  StringBuffer response = new StringBuffer();
  try {
   if (conn.getResponseCode() >= 400) {
    is = conn.getErrorStream();
   } else {
    is = conn.getInputStream();
   }
```

```
rd = new BufferedReader(new InputStreamReader(is));
   String line;
   while ((line = rd.readLine()) != null) {
    response.append(line);
    response.append('\n');
   }
  } catch (IOException ioe) {
   response.append(ioe.getMessage());
  } finally {
   if (is != null) {
    try {
     is.close();
    } catch (Exception e) {
    }
   }
   if (rd != null) {
    try {
     rd.close();
    } catch (Exception e) {
    }
   }
  }
  return (response.toString());
 }

  public static void main(String[] args) throws Exception {
    String baseUrl = "http://<hostname>:
<port>/webservices/rest/locator";
    String svcUrlStr1 = baseUrl + "/getRestInterface";
    //Invoke REST service
    invokeREST(svcUrlStr1, "<EBS_username>", "<password>", "PLSQL:
FND_PROFILE");
 }
}
```

Please note that resource information recorded earlier from the deployed WADL is now placed in the `baseUrl` and `svcUrlStr1` elements. PLSQL:FND_PROFILE is the interface internal name and that it is assumed to be deployed in the instance.

> **Note:** Use **https** (instead of http) in the `baseUrl` if your Oracle E-Business Suite instance is running on the SSL-enabled environment. Additionally, you need to import the SSL certificate into your client JVM's keystore.

10. Replace `<hostname>:<port>`, `<EBS_username>`, and `<password>` with the actual values in the code.

11. Save your work by selecting **File > Save All**.

12. **Add required library for JSON format:**

    Use the following steps to add the required library file to the project properties.

    1. Select and right-click on the project name you just created earlier to open a selection menu.

    2. Select **Project Properties** from the menu.

The Default Properties dialog box opens.

3. Select **Libraries and Classpath**, and click **Add Library**. The Add Library dialog box opens.

4. In the Add Library dialog box, select the Project folder and then click **New**.

   The Create Library dialog box opens.

5. In the Library Name field, enter 'Jersey-bundle_1.0.0.0_1-1-5-1.jar'.

   Click **Add Entry**. The Select Path Entry dialog box appears.

6. In the Select Path Entry dialog box, locate and select the 'Jersey-bundle_1.0.0.0 _1-1-5-1.jar' file that you have downloaded. This adds it to the Classpath.

   Click **OK**. The 'Jersey-bundle_1.0.0.0_1-1-5-1.jar' is now added to the Project folder.

7. The Project Properties dialog box appears. Click **OK**. This project is now set up with the required library.

Invoking a REST Service Using a Java Class

After creating a project with a Java class `RestInvocationGETMethod.java`, you need to compile and execute the process to invoke the REST service.

Use the following steps to compile and run the Java class:

1. In the Application Navigator, right-click on the `RestInvocationGETMethod.java` Java class you just created at the design time. Select **Make** from the menu.

2. Right-click on the `RestInvocationGETMethod.java` Java class. Select **Run** from the menu.

Monitor this process and check for successful compilation in the Log window. Verify that the execution is successful in the Log window.

Request Header Information

In this example, `getRestInterface` Java method is exposed as a REST service operation with the GET method. There is no input payload for the GET method. The path variable `{irepClassName}` is replaced with actual value "PLSQL: FND_PROFILE" sent as part of the HTTP URL shown below when the `getRestInterface` REST service operation is invoked.

```
URL = http://<hostname>:
<port>/webservices/rest/locator/getRestInterface/PLSQL:
FND_PROFILE
```

Note: For GET requests, JSON is the default output response format. Use `Accept` header `application/xml` to receive response in XML format. If `Content-Type` header is sent in a GET HTTP request, it will be ignored.

```
Request Headers
           Authorization: Basic xxxxxxxxxxxxxxxxxxxxxxx
           Accept: application/json
           Content-Language: en-US
```

Viewing Output Message

When the REST service is successfully invoked, the following output in JSON format appears in the Log window:

```
   {
                  "OutputParameters" : {
                    "EbsRestServiceBean" : [ {
                   "alternateAlias" : "plsql/PLSQL:FND_PROFILE",
                   "serviceAlias" : "profile",
                   "serviceName" : "PLSQL:FND_PROFILE",
                   "wadlUrl" : "http://<hostname>:
<port>/webservices/rest/profile?WADL"
                    } ],
                "ControlBean" : [ {
                      "fields" : "",
                      "filter" : "",
                      "limit" : "",
                      "offset" : ""
                } ]
          }
        }
```

Notice that the service information identified by its internal name `PLSQL:FND_PROFILE` is returned. For example, the WADL URL, service name, and service alias are included as part of the response message.

Annotating and Invoking a Custom Java Bean Service

Since not all Java APIs registered in the Integration Repository can be exposed as REST services, only some specialized Java APIs described earlier with proper annotation can be exposed as REST services. To better understand how to annotate those specialized Java APIs as Java Bean Services, and how to invoke Java Bean Services with REST service security, this section describes the entire process from annotating a custom Java API to invoking the service using the HTTP GET method.

REST Service Invocation Scenario

This example uses a custom Java API called Employee Service (`oracle.apps.per.sample.service.EmployeeInfo`) to explain the entire annotation and upload processes as well as the service invocation. There are two invocation scenarios. One scenario is to get employee details, and the other one is to get direct reports for the logged in user.

To get employee details, an HTML page with Javascript is used to make an HTTP GET

request to the `getPersonInfo` service operation contained in the custom API. An employee ID is provided at run time for the service to retrieve the associated employee name and the employee's manager name. After the successful service invocation, the employee details corresponding to the employee ID are displayed in the HTML page.

To get direct reports for the logged in user, a different HTML page with Javascript is used in this scenario to make an HTTP GET request to the `getDirectReports` service operation. The logged in user credentials are provided when the `getDirectReports` service operation is invoked. After the successful service invocation, the logged in user's subordinates or the user's direct reports are displayed in the HTML page with the requested number of records shown in each page.

High Level Process Flow for Creating Custom Java Bean Services

To develop custom Java Bean Services, an integration repository developer needs to create and annotate the custom Java APIs based on the Integration Repository Annotation Standards for Java Bean Services. After the interface creation, an integration repository administrator needs to validate the annotated APIs. If no error occurs during the validation, the administrator will then upload the custom APIs to Oracle Integration Repository where they can be published as REST services through Oracle E-Business Suite Integrated SOA Gateway.

For annotation information, see: Annotations for Java Bean Services, page A-6.

Annotating and Invoking a Custom Java Bean Service from HTML Pages with Javascript

Based on the REST service invocation scenario, the following tasks are included in this section:

Creating and Compiling Custom Java APIs

This section describes how to create and annotate a custom Java Bean Service called Employee Information (`oracle.apps.per.sample.service.EmployeeInfo`).

Guidelines for Developing Custom Java APIs

During the planning stage, use the following guidelines to plan and develop the custom

Java APIs that will be exposed as REST services through Oracle E-Business Suite Integrated SOA Gateway:

1. Develop a Java class whose public methods provide business functionality. Business logic should be embedded into these public methods.

   - The custom Java APIs and method names will be used in Web service URL. Therefore, ensure to provide friendly names.

   - If you need initialized Oracle E-Business Suite Context within the Java method, ensure the following:

     - Use ISG Context (IContext) to get handle to runtime information.

       For example, `IContext ctx = ContextManager.getContext();`

     - Retrieve fully initialized Oracle E-Business Suite WebAppsContext based on the request header / security token.

       For example, `WebAppsContext wctx = (WebAppsContext) ctx.getExternalContext();`

       `WebAppsContext` may be used later for application specific validation, such as fine grained access control.

   - You may use the Oracle E-Business Suite Integrated SOA Gateway database connections within the Java APIs.

     For example, `conn = DBConnectionManager.getConnection();`

   - Oracle E-Business Suite Integrated SOA Gateway provides a standard exception handling from the infrastructure. Ensure that the Java APIs throw or rethrow a throwable exception whenever an error condition occurs.

2. If above Java methods require complex data objects to be exchanged as input and output parameters, then develop Java Beans.

   Java Beans should:

   - Implement `java.io.Serializable`.

   - Have no-argument constructor.

   - Have accessor methods, following 'get' and 'set' naming convention, for private attributes.

Creating Custom Java APIs

In this example, you need a service that will return the details of a specific person in the hierarchy of the logged in user, as well as return all the reports of the logged in user.

To achieve this goal, create the following Java files:

- A Java class `EmployeeInfo.java` contains the following three methods. Based on the scenarios, they are read-only methods and we will map each Java method to HTTP GET verb.
  - `getAllReports` - This method returns an array of all reports of the requesting user.
  - `getDirectReports` - This method returns a list of direct reports of the requesting user.
  - `getPersonInfo` - This method returns the person details for a specific person Id.

    This method requires a key identifier parameter for "Person Id". Therefore, `personId` will be annotated as `key_param`.

    ```
    ...
    * @param personId Person Identifier
    * @rep:paraminfo {@rep:required} {@rep:key_param}
    ...
    ```

- A Java class `PerServiceException.java` that extends SOAException in Oracle E-Business Suite Integrated SOA Gateway.
- A Java Bean `PersonBean.java` to capture person information.

To create and compile custom Java APIs:

Use the following steps to create and compile custom Java APIs:

1. Create the first Java class called `PersonBean.java` with the following Java code:

   This Java class extends `java.lang.Ojbect`.

```
package oracle.apps.per.sample.beans;

import java.io.Serializable;

import java.util.Date;

public class PersonBean implements Serializable {

  private static final long serialVersionUID = 1L;

  private int personId;

  private String firstName;

  private String lastName;

  private String salutaion;

  private String fullName;

  private String nameSuffix;

  private String emailAddress;

  private String employeeNumber;

  private String workPhone;

  private Date startDate;

  private int reportingLevel;

  private int supervisorPersonId;

  private String supervisorEmployeeNumber;

  private String supervisorFullName;

  private String reportingHierarchy;


  public void setPersonId(int personId) {
    this.personId = personId;
  }

  public int getPersonId() {
    return personId;
  }

  public void setFirstName(String firstName) {
    this.firstName = firstName;
  }

  public String getFirstName() {
    return firstName;
  }

  public void setLastName(String lastName) {
    this.lastName = lastName;
  }

  public String getLastName() {
    return lastName;
  }
```

```
public void setSalutaion(String salutaion) {
    this.salutaion = salutaion;
  }

  public String getSalutaion() {
    return salutaion;
  }

  public void setFullName(String fullName) {
    this.fullName = fullName;
  }

  public String getFullName() {
    return fullName;
  }

  public void setEmailAddress(String emailAddress) {
    this.emailAddress = emailAddress;
  }

  public String getEmailAddress() {
    return emailAddress;
  }

  public void setEmployeeNumber(String employeeNumber) {
    this.employeeNumber = employeeNumber;
  }

  public String getEmployeeNumber() {
    return employeeNumber;
  }

  public void setStartDate(Date startDate) {
    this.startDate = startDate;
  }

  public Date getStartDate() {
    return startDate;
  }

  public void setNameSuffix(String nameSuffix) {
    this.nameSuffix = nameSuffix;
  }

  public String getNameSuffix() {
    return nameSuffix;
  }

  public void setWorkPhone(String workPhone) {
    this.workPhone = workPhone;
  }

  public String getWorkPhone() {
    return workPhone;
  }

  public void setReportingLevel(int reportingLevel) {
    this.reportingLevel = reportingLevel;
  }

  public int getReportingLevel() {
    return reportingLevel;
  }

  public void setSupervisorPersonId(int supervisorPersonId) {
```

```
this.supervisorPersonId = supervisorPersonId;
  }

  public int getSupervisorPersonId() {
    return supervisorPersonId;
  }

  public void setSupervisorEmployeeNumber(String
supervisorEmployeeNumber) {
    this.supervisorEmployeeNumber = supervisorEmployeeNumber;
  }

  public String getSupervisorEmployeeNumber() {
    return supervisorEmployeeNumber;
  }

  public void setSupervisorFullName(String supervisorFullName) {
    this.supervisorFullName = supervisorFullName;
  }

  public String getSupervisorFullName() {
    return supervisorFullName;
  }

  public void setReportingHierarchy(String reportingHierarchy) {
    this.reportingHierarchy = reportingHierarchy;
  }

  public String getReportingHierarchy() {
    return reportingHierarchy;
  }

}
```

2. Create the second Java class called `PerServiceException.java` with the following Java code:

   This Java class extends `java.lang.Ojbect`.

```
package oracle.apps.per.sample.common;

import oracle.apps.fnd.soa.util.SOAException;

public class PerServiceException extends SOAException {

  public PerServiceException(String message) {
      super(message);
  }

  public PerServiceException(Throwable t) {
      super(t);
  }

  public PerServiceException(String message, Throwable t) {
      super(message, t);
  }

  public PerServiceException(String errorCode, String message,
Throwable t) {
      super(errorCode, message, t);
  }

}
```

3. Create the third Java class called `EmployeeInfo.java` with the following Java code:

   This Java class extends `java.lang.Ojbect`.

```
package oracle.apps.per.sample.service;

import java.sql.Connection;
import java.sql.SQLException;

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

import oracle.apps.fnd.common.WebAppsContext;
import oracle.apps.fnd.isg.common.IContext;
import oracle.apps.fnd.isg.common.mgr.ContextManager;
import oracle.apps.fnd.isg.common.mgr.DBConnectionManager;
import oracle.apps.per.sample.beans.PersonBean;
import oracle.apps.per.sample.common.PerServiceException;

import oracle.jdbc.OraclePreparedStatement;
import oracle.jdbc.OracleResultSet;


/**
 * A sample class to demonstrate how Java API can use the ISG REST
framework. This class provides
 * methods to retrieve list of direct reports, all reports of a
person. It also has methods to
 * retrieve personal details and accrual balance of a person.
 * @rep:scope public
 * @rep:product PER
 * @rep:displayname Employee Information
 * @rep:category IREP_CLASS_SUBTYPE JAVA_BEAN_SERVICES
 */
public class EmployeeInfo {

  public EmployeeInfo() {
    super();
  }

  /**
     * This method returns a list of direct reports of the
requesting user.
     *
     * @return List of person records who are direct reports
     * @rep:paraminfo {@rep:innertype oracle.apps.per.sample.beans.
PersonBean}
     * @rep:scope public
     * @rep:displayname Get Direct Reports
     * @rep:httpverb get
     * @rep:category BUSINESS_ENTITY sample
     */
  // Demonstration of list return type
 public List<PersonBean> getDirectReports() throws
PerServiceException {

    // Get the ISG context, which has runtime information
    IContext ctx = ContextManager.getContext();
    // Retrieve fully initialized webappscontext, as per the request
header / security token
    WebAppsContext wctx = (WebAppsContext) ctx.getExternalContext();

    // Use webappscontext for apps specific validation e.g. fine
grained access control etc.
    int userId = wctx.getUserId();
    String userName = wctx.getUserName().toUpperCase();
```

```
// Here we intend to filter based on the specific user who is
invoking the service
    String filter = " AND LEVEL = 2 START WITH USER_NAME = :1";

    List<PersonBean> personList = null;
    try {
      Map<Integer, PersonBean> map = makeBean(filter, userName);
      if (map != null)
        personList = new ArrayList<PersonBean>(map.values());
    } catch (SQLException sqle) {
      throw new PerServiceException("SQL error while getting the all
reports", sqle);
    }

    return(personList);
  }
/**
     * This method returns an array of all reports of the requesting
user.
     *
     * @return Array of person records who are reporting into the
requesting user's organization hierarchy
     * @rep:scope public
     * @rep:displayname Get All Reports
     * @rep:httpverb get
     * @rep:category BUSINESS_ENTITY sample
     */
  // Demonstration of array return type
 public PersonBean[] getAllReports() throws PerServiceException {

    // Get the ISG context, which has runtime information
    IContext ctx = ContextManager.getContext();
    // Retrieve fully initialized webappscontext, as per the request
header / security token
    WebAppsContext wctx = (WebAppsContext) ctx.getExternalContext();


    // Use webappscontext for apps specific validation e.g. fine
grained access control etc.
    int userId = wctx.getUserId();
    String userName = wctx.getUserName().toUpperCase();

    // Here we intend to filter based on the specific user who is
invoking the service
    String filter = " START WITH USER_NAME = :1 ";

    PersonBean[] array = null;

    try {
      Map<Integer, PersonBean> map = makeBean(filter, userName);
      if (map != null)
        array = map.values().toArray(new PersonBean[map.size()]);
    } catch (SQLException sqle) {
      throw new PerServiceException("SQL error while getting the
direct reports", sqle);
    }

    return(array);

  }

  /**
   * This method returns the person details for a specific person
id. Throws error if the person
   * is not in requesting user's org hierarchy.
```

```
*
   * @return Details of a person in the logged on user's org
hierarchy.
   * @param personId Person Identifier
   * @rep:paraminfo {@rep:required} {@rep:key_param}
   * @rep:scope public
   * @rep:displayname Get Person Details
   * @rep:httpverb get
   * @rep:category BUSINESS_ENTITY sample
   */
  // Demonstration of simple navigation using path param
  public PersonBean getPersonInfo(int personId) throws
PerServiceException {

    // Get the ISG context, which has runtime information
    IContext ctx = ContextManager.getContext();
    // Retrieve fully initialized webappscontext, as per the request
header / security token
    WebAppsContext wctx = (WebAppsContext) ctx.getExternalContext();

    // Use webappscontext for apps specific validation e.g. fine
grained access control etc.
    int userId = wctx.getUserId();
    String userName = wctx.getUserName().toUpperCase();

    String filter = " START WITH USER_NAME = :1 ";
    Map<Integer, PersonBean> map = null;
    try {
      map = makeBean(filter, userName);
    } catch (SQLException sqle) {
      throw new PerServiceException("SQL error while getting the
direct reports", sqle);
    }

    if (map == null)
      throw new PerServiceException(PerServiceException.
AUTHORIZATION_FAILURE, "No org hierarchy found for user - " +
userName, null);

    boolean doesExist = map.containsKey(personId);
    if (!doesExist)
      throw new PerServiceException(PerServiceException.
AUTHORIZATION_FAILURE, "The given person " + personId + " either
does not exist or does not belong to the current user's - " +
userName + " hierarchy", null);

    PersonBean bean = map.get(personId);

    return(bean);

  }

  /************************ private members
***************************/

  private Map<Integer, PersonBean> makeBean(String sqlFilter, String
userName) throws SQLException {

    Connection conn = null;
    OraclePreparedStatement stmt = null;
    OracleResultSet rs = null;
    HashMap<Integer, PersonBean> personMap = new HashMap<Integer,
PersonBean>();
    IContext ctx = ContextManager.getContext();
```

```
try {

      String sql = GET_PERSON_INFO1 + sqlFilter;

      // preferred way of obtaining connection, rather than from
WebAppsContext
      conn = DBConnectionManager.getConnection();
      stmt = (OraclePreparedStatement) conn.prepareStatement(sql);
      stmt.setString(1, userName);
      rs = (OracleResultSet) stmt.executeQuery();

      while (rs != null && rs.next()) {

        int personId = rs.getInt(1);
        String firstName = rs.getString(2);
        String lastName = rs.getString(3);
        String title = rs.getString(4);
        String fullName = rs.getString(5);
        String nameSuffix = rs.getString(6);
        String empNumber = rs.getString(7);
        String emailAddr = rs.getString(8);
        String workPhone = rs.getString(9);
        int supervisorId = rs.getInt(10);
        java.util.Date startDate = rs.getDate(12);
        int level = rs.getInt(16);
        String supervisorName = rs.getString(17);
        String supervisorEmpNumber = rs.getString(18);
        String reporting = rs.getString(19);

        // Do any data validation / transformation if necessary,
here

        PersonBean personBean = new PersonBean();

        // Set the information into the serializable bean
        personBean.setPersonId(personId);
        personBean.setFirstName(firstName);
        personBean.setLastName(lastName);
        personBean.setSalutaion(title);
        personBean.setFullName(fullName);
        personBean.setNameSuffix(nameSuffix);
        personBean.setEmployeeNumber(empNumber);
        personBean.setEmailAddress(emailAddr);
        personBean.setWorkPhone(workPhone);
        personBean.setSupervisorPersonId(supervisorId);
        personBean.setStartDate(startDate);
        personBean.setReportingLevel(level);
        personBean.setSupervisorFullName(supervisorName);
        personBean.setSupervisorEmployeeNumber(supervisorEmpNumber);
        personBean.setReportingHierarchy(reporting);

        personMap.put(personId, personBean);

      }

    } finally {

      if (rs != null) {
        try { rs.close(); } catch (Exception e) {};
        rs = null;
      }

      if (stmt != null) {
        try { stmt.close(); } catch (Exception e) {};
        stmt = null;
```

```
}

      // preferred way of closing the connection
      DBConnectionManager.closeConnection(conn);

    }

    return(personMap);

  }

  private static final String GET_PERSON_INFO1 = "SELECT  DISTINCT
E.PERSON_ID, E.FIRST_NAME, E.LAST_NAME, E.TITLE, E.FULL_NAME, E.
SUFFIX,\n" +
                                               "E.EMPLOYEE_NUMBER, E.
EMAIL_ADDRESS, E.WORK_TELEPHONE, E.SUPERVISOR_ID,\n" +
                                               "E.
SUPERVISOR_ASSIGNMENT_ID, E.EFFECTIVE_START_DATE, E.
EFFECTIVE_END_DATE,\n" +
                                               "E.USER_ID, E.USER_NAME,
LEVEL, PRIOR E.FULL_NAME, PRIOR E.EMPLOYEE_NUMBER,\n" +
                                               "SYS_CONNECT_BY_PATH(E.
FULL_NAME, '/')\n" +
                                               "FROM\n" +
                                               "(SELECT PPF.PERSON_ID,\n"
+
                                               "PPF.FIRST_NAME,\n" +
                                               "PPF.LAST_NAME,\n" +
                                               "PPF.TITLE,\n" +
                                               "PPF.FULL_NAME,\n" +
                                               "PPF.SUFFIX,\n" +
                                               "PPF.EMPLOYEE_NUMBER,\n" +
                                               "PPF.EMAIL_ADDRESS,\n" +
                                               "PPF.WORK_TELEPHONE,\n" +
                                               "PAF.SUPERVISOR_ID,\n" +
                                               "PAF.
SUPERVISOR_ASSIGNMENT_ID,\n" +
                                               "GREATEST(PPF.
EFFECTIVE_START_DATE,    PAF.EFFECTIVE_START_DATE)
EFFECTIVE_START_DATE, \n" +
                                               "LEAST(PPF.
EFFECTIVE_END_DATE,    PAF.EFFECTIVE_END_DATE) EFFECTIVE_END_DATE,
\n" +
                                               "FUR.USER_ID,\n" +
                                               "FUR.USER_NAME\n" +
                                               "FROM FND_USER FUR,
PER_ALL_PEOPLE_F PPF,\n" +
                                               "PER_ALL_ASSIGNMENTS_F
PAF\n" +
                                               "WHERE PPF.PERSON_ID =
PAF.PERSON_ID (+)\n" +
                                               "AND PPF.BUSINESS_GROUP_ID
= PAF.BUSINESS_GROUP_ID (+)\n" +
                                               "AND (SYSDATE BETWEEN PPF.
EFFECTIVE_START_DATE\n" +
                                               "AND PPF.
EFFECTIVE_END_DATE\n" +
                                               "AND SYSDATE BETWEEN PAF.
EFFECTIVE_START_DATE\n" +
                                               "AND PAF.
EFFECTIVE_END_DATE)\n" +
                                               "AND FUR.EMPLOYEE_ID =
PPF.PERSON_ID\n" +
                                               ") E\n" +
                                               "CONNECT BY SUPERVISOR_ID
```

```
= PRIOR PERSON_ID";

}
```

4. Place these Java class files in the respective folders in the JAVA_TOP directory.

5. Navigate to the respective folders in JAVA_TOP and execute the corresponding commands to compile Java class files:

   ```
   java PersonBean.java
   java PerServiceException.java
   java EmployeeInfo.java
   ```

   The corresponding .class files will be created in the respective folders in the JAVA_TOP directory.

6. Stop and restart Oracle E-Business Suite application server.

Parsing and Uploading the Annotated Custom Java Bean Service to the Integration Repository

Once the custom Java classes have been successfully deployed to a target instance, the integration repository administrator needs to validate the annotated custom interface definition Employee Information against the annotation standards for Java Bean Services using Integration Repository Parser. During the validation, if no error occurs, an iLDT file will be generated. The administrator will then upload the generated iLDT file to Oracle Integration Repository.

Perform the following steps to parse and upload the annotated custom interface definition Employee Information to Oracle Integration Repository:

1. Source the environment in the run file system and set CLASSPATH to include all libraries and JAR files used by the custom Java API.

2. Ensure the annotated custom interface definition and related Java classes are located in the target instance.

3. Execute the Integration Repository Parser using the following command to validate the annotated custom interface definition against the annotation standards:

   ```
   $IAS_ORACLE_HOME/perl/bin/perl $FND_TOP/bin/irep_parser.pl -g
   -v -username=sysadmin per:patch/115/java:EmployeeInfo.java:
   12.1=EmployeeInfo.java
   ```

Integration Repository Parser Command Process and Validation

```
-bash-3.2$ $IAS_ORACLE_HOME/perl/bin/perl $FND_TOP/bin/irep_parser.pl -g -v -username=sysadmin per:patch/115/java:EmployeeIn
fo.java:12.1=EmployeeInfo.java
# Interface Repository Annotation Processor, 12.0.0

#
# Generating annotation output.
# Processing file 'EmployeeInfo.java'.
# Using YAPP-based parser.
# Warning: Carrige return(s) seen in text.
# Java code is in package 'oracle.apps.per.sample.service'.
# Defining the class 'EmployeeInfo'.
# Found a class annotation for 'EmployeeInfo'.
# Found a member annotation for the method named getDirectReports
# Found a member annotation for the method named getAllReports
# Found a member annotation for the method named getPersonInfo
# Javadoc at line 158 with no IR annotation tags ignored.
# Done all files.
-bash-3.2$ ll
total 100
-rw-r--r-- 1 oracle dba  8358 Jul  3 11:20 EmployeeInfo.class
-rw-r--r-- 1 oracle dba 11463 Jul  3 11:21 EmployeeInfo.java
-rw-r--r-- 1 oracle dba 14765 Jul  3 11:34 EmployeeInfo_java.ildt
-rw-r--r-- 1 oracle dba  1281 Jul  3 04:01        .log
-rw-r--r-- 1 oracle dba 15191 Jul  2 05:36 PersonResource.class
-rw-r--r-- 1 oracle dba 19311 Jul  2 23:33 PersonResource.java
-rw-r--r-- 1 oracle dba 17587 Jul  3 03:51 PersonResource_java.ildt
-bash-3.2$ clear
```

Notice that the `EmployeeInfo_java.ildt` file is successfully generated after the validation.

4. Upload the generated `EmployeeInfo_Java.ildt` file to the Integration Repository by using the following FNDLOAD command:

   ```
   $FND_TOP/bin/FNDLOAD apps/password@TWO_TASK 0 Y UPLOAD
   $FND_TOP/patch/115/import/wfirep.lct EmployeeInfo_java.ildt
   ```

5. Verify the generated log file to view the upload details.

Upload Details in the Log File

```
+---------------------------------------------------------------------------+
Application Object Library: Version : 12.1

Copyright (c) 1998, 2013, Oracle and/or its affiliates. All rights reserved.

FNDLOAD: Generic Loader
+---------------------------------------------------------------------------+

Current system time is 03-JUL-2014 11:36:46

+---------------------------------------------------------------------------+

Uploading from the data file EmployeeInfo_java.ildt
Altering database NLS_LANGUAGE environment to AMERICAN
Dump from LCT/LDT files (/u01/R121_EBS/fs1/EBSapps/appl/fnd/12.0.0/patch/115/import/wfirep.lct(120.8.12020000.3), EmployeeIn
fo_java.ildt) to stage tables
Dump LCT file /u01/R121_EBS/fs1/EBSapps/appl/fnd/12.0.0/patch/115/import/wfirep.lct(120.8.12020000.3) into FND_SEED_STAGE_CO
NFIG
Dump LDT file EmployeeInfo_java.ildt into FND_SEED_STAGE_ENTITY
Dumped the batch (IREP_OBJECT JAVA:oracle.apps.per.sample.service.EmployeeInfo C , PARAMS 0 0 ) into FND_SEED_STAGE_ENTITY
Upload from stage tables

+---------------------------------------------------------------------------+
Concurrent request completed successfully
Current system time is 03-JUL-2014 11:36:52

+---------------------------------------------------------------------------+
~
```

Notice that the upload process has been completed successfully.

6. Search the uploaded custom Java Service Beans interface "Employee Information" from the Integration Repository.

   Click the Employee Information name link to open the interface details page.

Deploying a Custom Java Bean Service

Use the following steps to deploy the custom interface as a REST service:

1. Log in to Oracle E-Business Suite as a user who has the Integration Repository Administrator role.

   Select the Integrated SOA Gateway responsibility and the Integration Repository link from the navigation menu.

2. In the Integration Repository tab, click **Search** to access the main Search page.

3. Click **Show More Search Options** to display more search fields.

   Select 'Custom' in the Interface Source field.

4. Click **Go** to execute the search.

   Click the "Employee Information" interface name link to open the interface details page.

5. In the REST Web Service tab, enter the following information:

   - Service Alias: empinfo

     The alias will be displayed as the service endpoint in the WADL and schema for the selected method or operation.

   - Select the POST check box for the Get Person Details Java method. Leave the rest of preselected check boxes unchanged.

6. Click **Deploy** to deploy the service to an Oracle E-Business Suite environment.

Once the REST service has been successfully deployed, 'Deployed' appears in the REST Service Status field along with the **View WADL** link. Click the **View WADL** link to view the deployed service WADL description.

For more information on deploying REST services, see Deploying REST Web Services, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

Creating a Security Grant

After deploying the custom interface "Employee Information" as a REST service, the integration repository administrator can create a security grant to authorize the service access privileges to all users.

Use the following steps to create a security grant:

1. Log in to Oracle E-Business Suite as a user who has the Integration Repository Administrator role. Select the Integrated SOA Gateway responsibility and the

Integration Repository link from the navigation menu.

2. Perform a search to locate the "Employee Information" service the administrator just deployed earlier.

3. Select the "Employee Information" name link from the search result table to display the interface details page. Click the Grants tab.

4. Select the Employee Information service and then click **Create Grant**.

5. In the Create Grants page, select "All Users" as the Grantee Type.

   Note: In this example, the `Employee Information` service is granted to all users. In actual implementation, you should define strict security rules. Create grant to a user or more appropriately to a group of users defined by roles and responsibilities.

   Click **Create Grant**. This grants the selected service access privilege to all Oracle E-Business Suite users.

Recording Resource Information from Deployed WADL

To obtain service resource information from the deployed WADL for the custom Employee Information service, click the **View WADL** link in the REST Web Service tab.

The following WADL description appears:

```
<xml version="1.0" encoding="UTF-8">
<application name="EmployeeInfo" targetNamespace="http://xmlns.oracle.
com/apps/per/soaprovider/pojo/employeeinfo/"
 xmlns:tns="http://xmlns.oracle.
com/apps/per/soaprovider/pojo/employeeinfo/"
 xmlns="http://wadl.dev.java.net/2009/02" xmlns:xsd="http://www.w3.
org/2001/XMLSchema"
 xmlns:tns1="http://xmlns.oracle.
com/apps/fnd/rest/empinfo/getallreports/"
 xmlns:tns2="http://xmlns.oracle.
com/apps/fnd/rest/empinfo/getdirectreports/"
 xmlns:tns3="http://xmlns.oracle.
com/apps/fnd/rest/empinfo/getpersoninfo/">

<grammars>
    <include href="http://<hostname>:<port>/webservices/rest/empinfo/?
XSD=getallreports.xsd" xmlns="http://www.w3.org/2001/XMLSchema" />
   <include href="http://<hostname>:<port>/webservices/rest/empinfo/?
XSD=getdirectreports.xsd" xmlns="http://www.w3.org/2001/XMLSchema" />
   <include href="http://<hostname>:<port>/webservices/rest/empinfo/?
XSD=getpersoninfo.xsd" xmlns="http://www.w3.org/2001/XMLSchema" />
</grammars><resources base="http://<hostname>:
<port>/webservices/rest/empinfo/">
  <resource path="/getAllReports/">
 <method id="getAllReports" name="GET">
     <request>
     <param name="ctx_responsibility" type="xsd:string" style="query"
required="false" />
          <param name="ctx_respapplication" type="xsd:string" style="
query" required="false" />
    <param name="ctx_securitygroup" type="xsd:string" style="query"
required="false" />
    <param name="ctx_nlslanguage" type="xsd:string" style="query"
required="false" />
    <param name="ctx_orgid" type="xsd:int" style="query" required="
false" />
   </request>
   <response>
     <representation mediaType="application/xml" type="tns1:
getAllReports_Output" />
     <representation mediaType="application/json" type="tns1:
getAllReports_Output" />
    </response>
   </method>
  </resource>
 <resource path="/getDirectReports/">
  <method id="getDirectReports" name="GET">
    <request>
     <param name="ctx_responsibility" type="xsd:string" style="query"
required="false" />
             <param name="ctx_respapplication" type="xsd:string"
style="query" required="false" />
     <param name="ctx_securitygroup" type="xsd:string" style="query"
required="false" />
     <param name="ctx_nlslanguage" type="xsd:string" style="query"
required="false" />
     <param name="ctx_orgid" type="xsd:int" style="query" required="
false" />
    </request>
      <response>
     <representation mediaType="application/xml" type="tns2:
getDirectReports_Output" />
     <representation mediaType="application/json" type="tns2:
getDirectReports_Output" />
    </response>
   </method>
```

```
</resource>
 <resource path="="/getPersonInfo/ {personId}/">
  <param name="personId" style="template" required="true" type="xsd:int"
/>
 <method id="getPersonInfo" name="GET">
     <request>
     <param name="ctx_responsibility" type="xsd:string" style="query"
required="false" />
          <param name="ctx_respapplication" type="xsd:string" style="
query" required="false" />
    <param name="ctx_securitygroup" type="xsd:string" style="query"
required="false" />
    <param name="ctx_nlslanguage" type="xsd:string" style="query"
required="false" />
    <param name="ctx_orgid" type="xsd:int" style="query" required="
false" />
   </request>
   <response>
     <representation mediaType="application/xml" type="tns3:
getPersonInfo_Output" />
     <representation mediaType="application/json" type="tns3:
getPersonInfo_Output" />
    </response>
   </method>
  </resource>
 <resource path="/getPersonInfo/">
  <method id="getPersonInfo" name="POST">
    <request>
       <representation mediaType="application/xml" type="tns3:
getPersonInfo_Input" />
     <representation mediaType="application/xml" type="tns3:
getPersonInfo_Output" />
    </request>
      <response>
       <representation mediaType="application/xml" type="tns3:
getPersonInfo_Input" />
     <representation mediaType="application/xml" type="tns3:
getPersonInfo_Output" />    </response>
   </method>
  </resource>
  </resource path>
</application>
```

Copy or record the following information which will be used later when defining a Java client:

- `<resources base>="http://<hostname>:<port>/webservices/rest/empinfo/">`

  This information `http://<hostname>:<port>/webservices/rest/empinfo` will be used later in Java client program as baseUrl.

- `<resource path>="/getPersonInfo/{personId}/">`

  This information `/getPersonInfo/{personId}/` will be used later to form the later part of the service URL.

  Note that **{personId}** is a path variable defined using the `<param>` tag. It will be replaced with actual value (employee ID 13137 in this example) at run time when the HTTP GET method is invoked.

- `<resource path="/getDirectReports/">`

This information `/getDirectReports/` will be used later to form the later part of the service URL.

Invoking a Custom REST Service from HTML Using Javascript

This example contains two invocation scenarios:

1. Get employee personal information by invoking the `getPersonInfo` REST service operation

2. Get direct reports for the logged in user by invoking the `getDirectReports` REST service operation with pagination control parameter

Scenario 1: Get Employee Personal Information

Use the following steps to invoke the deployed the `getPersonInfo` REST service operation:

1. Create an HTML file using any text editor with the following content:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
 <head>
  <title>Oracle E-Business Suite Integrated SOA Gateway - REST
Services Sample</title>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"
>
  <style>
  table,th,td
  {
   border:1px solid black;
   border-collapse:collapse;
  }
  th,td
  {
   padding:5px;
  }
  </style
  <script language="javascript">
   function getEmpDetails()
  {
   var xmlhttp;
   if (window.XMLHttpRequest)
   {// code for IE7+, Firefox, Chrome, Opera, Safari
    xmlhttp=new XMLHttpRequest();
   }
   else
   {// code for IE6, IE5
    xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
   }
   xmlhttp.onreadystatechange=function()
   {
    if (xmlhttp.readyState==4 && xmlhttp.status==200)
    {
     var det = eval( "(" +  xmlhttp.responseText + ")");
     document.getElementById('div1').innerHTML=det.OutputParameters.
PersonBean[0].fullName;
     document.getElementById('div2').innerHTML=det.OutputParameters.
PersonBean[0].supervisorFullName;
    }
   }

   var empno = document.getElementById("empno");
   var url = "http://<hostname>:<port>/webservices/rest/empinfo/
getPersonInfo/" + empno.value;

   xmlhttp.open('GET',url,true,"cbrown", password);
   xmlhttp.send();
  }
 </script>
</head>
<body>
 <center><h1>Oracle E-Business Suite Integrated SOA Gateway - REST
Services Sample</h1>
        <h2>Path Parameter Example</h2>
 </center>
  <h3>This example demonstrates the use of Path Parameters in REST
Service URL.
 ISG REST Service expects personId as input. personId was annotated
as key_param. Hence, it can be sent as part of GET request URL.
 This example does not set Accept header in HTTP Request. REST
Service returns response in JSON format (JSON is default message
format).<br><br>
Enter employee id and click Get Details button. ISG REST Service
will be invoked. Employee Name and his / her Manager's Name will be
```

```
displayed.<br>
</h3>
 <center><table>
<tr>
  <td>Enter Employee ID</td>
  <td><input type="text" id="empno" size="10"/> <input type="button"
value="Get Details" onclick="getEmpDetails()"/>
</tr>
<tr>
  <td>Employee Name</td>
  <td><div id="div1/></div></td>
</tr>
<tr>
  <td>Manager Name</td>
  <td><div id="div1/></div></td>
</tr>
</table>
</body>
</html>
```

This invocation scenario uses path parameter **{personId}** to obtain the employee ID at run time. The `getPersonInfo` service operation is then invoked through the HTTP GET request.

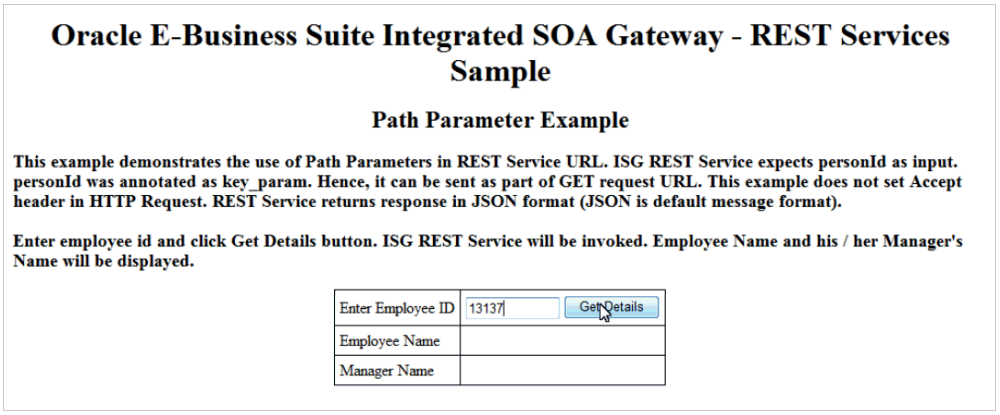
```
<resource path="="getPersonInfo/{personId}/">
  <param name="personId" style="template" required="true" type="xsd:
int"/>
    <method id="getPersonInfo" name="GET">
```

The path parameter **{personId}** was annotated earlier in `@rep:key_param` and can be sent as a GET request URL.

This example does not set the `Accept` header in the GET request. The REST service returns a response message in the default JSON format.

2. View the HTML page in a browser window.

HTML Page to Invoke a REST Service for Path Parameter Example



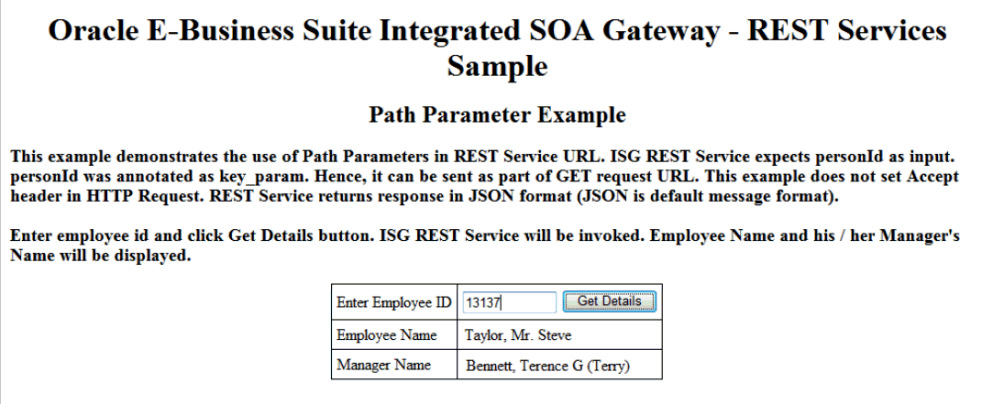
3. Enter an Employee ID (path parameter `{personId}`), such as 13137, in the HTML page and click **Get Details**. This invokes the `getPersonInfo` REST service operation and returns the associated employee's name and his or her manager's

name.

In this example, Taylor, Mr. Steve who has the employee ID 13137 is displayed as the Employee Name; Bennett, Terrence G (Terry) who is Taylor's manager is displayed as Manager Name.

> **Note:** Employee details in this example represent a fictitious sample (based upon made up data used in the Oracle Demo Vision instance). Any similarity to actual persons, living or dead, is purely coincidental and not intended in any manner.

HTML Page with Employee Personal Information



Scenario 2: Get Direct Reports for the Logged in User

Use the following steps to invoke the deployed the `getDirectReports` REST service operation:

1. Create an HTML file using any text editor with the following content:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
 <head>
  <title>Oracle E-Business Suite Integrated SOA Gateway - REST
Services Sample</title>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"
>
  <style>
  table,th,td
  {
   border:1px solid black;
   border-collapse:collapse;
  }
  th,td
  {
   padding:5px;
  }
  </style
  <script language="javascript">
   function getPrev()
  {
   offset = parseInt(offset) - 3;
   getDirectReports(offset);
  }

  function getNext()
  {
   offset = parseInt(offset) + 3;
   getDirectReports(offset);
  }

  function getDirectReports(offsetP)
  {
   var xmlhttp;
   if (window.XMLHttpRequest)
   {// code for IE7+, Firefox, Chrome, Opera, Safari
    xmlhttp=new XMLHttpRequest();
   }
   else
   {// code for IE6, IE5
    xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
   }
   xmlhttp.onreadystatechange=function()
   {
    if (xmlhttp.readyState==4 && xmlhttp.status==200)
    {

     xmlDoc=xmlhttp.responseXML;
     txt="";
     retRows=0;
     x=xmlDoc.getElementsByTagName("PersonBean");

     if (x.length > 0)
     {
      txt = "<br><table border = \"1\"><tr><th>Employee
Id</th><th>Employee Name</th></tr>";
     }
     else
     {
      txt = "No records to display";
     }

     for (i=0;i<3;i++)
     {
      txt = txt + "<tr><td>" + xmlDoc.getElementsByTagName
```

```
("personId")[i].firstChild.nodeValue + "+ xmlDoc.
getElementsByTagName("personId")[i].firstChild.nodeValue + "</td>";
      txt = txt + "<td>" + xmlDoc.getElementsByTagName("fullName")
[i].firstChild.nodeValue + " + xmlDoc.getElementsByTagName
("fullName")[i].firstChild.nodeValue + "</td></tr>";
     }

     if (x.length > 0)
     {
      txt = txt + "</table><br>";
     }

     y=xmlDoc.getElementsByTagName("ControlBean");
     rOffset = xmlDoc.getElementsByTagName("offset")[0].firstChild.
nodeValue;
     rFrom = 1 + parseInt(rOffset);
     rTo = 3 + parseInt(rOffset);
     txt = txt + "Displaying " + rFrom;
     txt = txt + " to " + rTo + " records.<br>";
     document.getElementById("div1").innerHTML=txt;

     if(offsetP > 2) {
      document.getElementById("div2").innerHTML="<input type=\"
button\" value=\"Previous\" onclick=\"getPrev()\"/>";
     }
     else
     {
      document.getElementById("div2").innerHTML="";
     }

     if(x.length == 4) {
      document.getElementById("div3").innerHTML="<input type=\"
button\" value=\"Next\" onclick=\"getNext()\"/>";
     }
     else
     {
      document.getElementById("div3").innerHTML="";
     }


    }
   }

        var url = "http://<hostname>:
<port>/webservices/rest/empinfo/getDirectReports/?ctx_orgid=204&
limit=4&offset" + offsetP;

   xmlhttp.open('GET',url,true);
   xmlhttp.setRequestHeader("Accept", "application/xml");
   xmlhttp.send();
  }
 </script>
</head>
<body>
 <center><h1>Oracle E-Business Suite Integrated SOA Gateway - REST
Services Sample</h1>
         <h2>Pagination Control Parameters Example</h2>
 </center>
  <h3>This example demonstrates the use of Pagination Control
Parameters. When you click Get Directs button, ISG REST Service will
be invoked. Provide EBS username
 and password. On successful invocation, the first 3 direct reports
(employees) of the logged in user will be displayed. Next and
Previous buttons will be displayed
 based on the number of records.
```

```
</h3>
 <center>
  <input type="button" value="Get Directs" onclick="getDirectReports
(0)"/>
 <div id="div1/></div>
   <div id="div2/></div>
   <div id="div3/></div>
 </center>

<script type="text/javascript"><!--
        offset=0;
    //--></script>

</body>
</html>
```

This invocation scenario uses pagination control parameter **offset** defined in the Javascript to limit the number of records returned at run time and for pagination when `getDirectReports` service operation is invoked through HTTP GET request.

```
<script language="javascript">
...

        var url = "http://<hostname>:
<port>/webservices/rest/empinfo/getDirectReports/?ctx_orgid=204&
limit=4&offset" + offsetP;

   xmlhttp.open('GET',url,true);
   xmlhttp.setRequestHeader("Accept", "application/xml");
   xmlhttp.send();
  }
</script>
```

This example sets the `Accept` header in the GET request. The REST service returns a response message in XML format.

2. View the HTML page in a browser window.

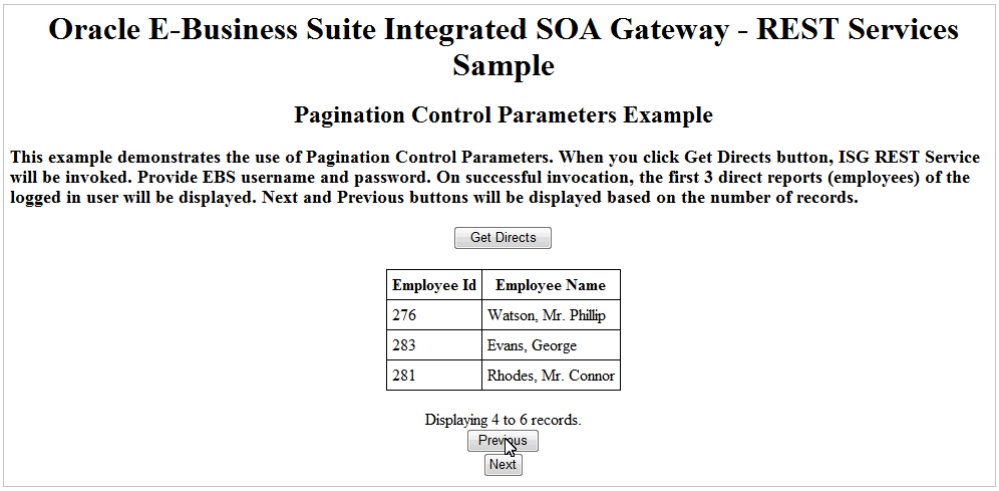
HTML Page to Invoke a REST Service for Pagination Control Parameters Example

3. Click **Get Directs** in the HTML page. This invokes the `getPersonInfo` REST service operation. A pop-up window appears requiring you to enter the username and password.

   After you entered the username and password and clicked **OK**, the service returns the first 3 direct reports (employees) of the user you just entered in the table. Click

Next to display the next 3 records retrieved from the direct reports list. Click **Previous** to display the last 3 records from the current direct reports list.

HTML Page with Direct Reports Shown as the Results



5

Using XML Gateway Inbound and Outbound Interfaces

Overview

Oracle E-Business Suite Integrated SOA Gateway provides a communication infrastructure between Oracle E-Business Suite and Web consumers. Inbound and outbound XML data is exchanged between the consumers and Oracle E-Business Suite through Oracle XML Gateway.

Oracle XML Gateway provides a common, standards-based approach for XML integration. XML is key to an integration solutions, as it standardizes the way in which data is searched, exchanged, and presented thereby enabling interoperability throughout the supply chain.

Oracle XML Gateway provides a set of services that can be easily integrated with Oracle E-Business Suite to support XML messaging. It uses the message propagation feature of Oracle Advanced Queuing to integrate with Oracle Transport Agent to deliver outbound XML messages to and receive inbound XML messages or transactions from business partners.

To enable bidirectional integration with Oracle E-Business Suite and consumers, Oracle E-Business Suite Integrated SOA Gateway supports XML Gateway Map interface type through the following approaches:

- For an inbound XML Gateway Map interface, once a Web service of an inbound XML Gateway interface is deployed, the deployed service representing in WSDL can be used in creating a BPEL process to insert inbound data into Oracle E-Business Suite.

- For an outbound XML Gateway Map interface, since an outbound message is first enqueued to the ECX_OUTBOUND queue, Oracle E-Business Suite Integrated SOA Gateway supports it through subscription model by first dequeuing the message to retrieve outbound data from Oracle E-Business Suite by using a BPEL process. The

retrieved data can then be passed to trading partners or consumers who subscribed to the message.

To better understand how to use a deployed Web service of an inbound XML Gateway interface as well as understand how the subscription model works for an outbound XML Gateway, the following topics are introduced in this chapter:

- Using XML Gateway Inbound Web Services, page 5-2
  - Using XML Gateway Inbound Services at Design Time, page 5-2
  - Deploying and Testing the BPEL Process at Run Time, page 5-23
- Using XML Gateway Outbound Through Subscription Model, page 5-27
  - Using XML Gateway Outbound Messages in Creating a BPEL Process at Design Time, page 5-27
  - Deploying and Testing a BPEL Process at Run Time, page 5-25

For the examples described in the following sections, we use Oracle JDeveloper 10.1.3.3.0 as a design-time tool to create the BPEL processes and use Oracle SOA Suite BPEL server 10.1.3.3.0 for the process deployment.

Using XML Gateway Inbound Services

This section includes the following topics:

- Using XML Gateway Inbound Services at Design Time, page 5-2
- Deploying and Testing the BPEL Process at Run Time, page 5-23

Using XML Gateway Inbound Services at Design Time

BPEL Process Scenario

Take the XML Gateway Inbound Process PO XML Transaction as an example to explain the BPEL process creation. In this example, the XML Gateway inbound message map is exposed as a web service through `PROCESS_PO_007` inbound map. It allows sales order data including header and line items to be inserted into Order Management system while an associated purchase order is created.

When a purchase order is sent by a trading partner, the purchase order data is used as input to the BPEL process along with ECX Header properties such as `MESSAGE_TYPE`, `MESSAGE_STANDARD`, `TRANSACTION_TYPE`, `TRANSACTION_SUBTYPE`, `PARTY_SITE_ID`, and `DOCUMENT_NUMBER`. The BPEL process then pushes this purchase order in ECX_INBOUND queue. Agent Listeners running on ECX_INBOUND would enable further processing by the Execution Engine. Oracle XML Gateway picks

this XML message, does trading partner validation, and inserts order data to Order Management Application.

If the BPEL process is successfully executed after deployment, you should get the same order information inserted into the Order Management table once a purchase order is created.

Prerequisites to Configure a BPEL Process Using an XML Gateway Inbound Service

Before performing the design-time tasks for XML Gateway Inbound services, you need to ensure the following tasks are in place:

- An integration repository administrator needs to successfully deploy the XML Gateway Inbound message map to the application server.
- An integration developer needs to locate and record the deployed WSDL URL for the inbound message map exposed as a web service.
- XML Gateway header variables need to be populated for XML transaction.
- Agent listeners need to be up and running.

Deploying XML Gateway Inbound WSDL URL

An integration repository administrator must first create a web service for the selected XML Gateway inbound map, and then deploy the service from Oracle Integration Repository to the application server.

For example, the administrator must perform the following steps before letting the integration developers use the deployed WSDL in creating a BPEL process:

1. To generate a web service, locate the interface definition first (such as an XML Gateway inbound interface `INBOUND:Process Purchase Order XML Transaction (ONT:POI)`) and click **Generate WSDL** in the interface details page.

   Once the service is successfully generated, the Web Service - SOA Provider region appears in the interface details page.

   > **Note:** Since XML Gateway Map interface types can be service enabled by Web Service Provider (for Oracle E-Business Suite Release 12.0) and SOA Provider (after the Release 12.0), you may also find the Web Service - Web Service Provider region as well if there are Web services generated through Web Service Provider from Oracle E-Business Suite Release 12.0.
   >
   > However, there is no design difference in terms of creating BPEL processes using XML Gateway inbound services whether the services are enabled through SOA Provider or Web Service Provider.

For detailed instruction on how to generate a web service, see Generating Web Services, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

2. To deploy a generated web service, select at least one authentication type and click **Deploy** in the Web Service - SOA Provider region of the interface details page to deploy the service.

   Once the service is successfully deployed, the selected authentication type(s) will be displayed along with 'Deployed' Web Service Status. For more information on securing web services with authentication types, see Managing Web Service Security, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

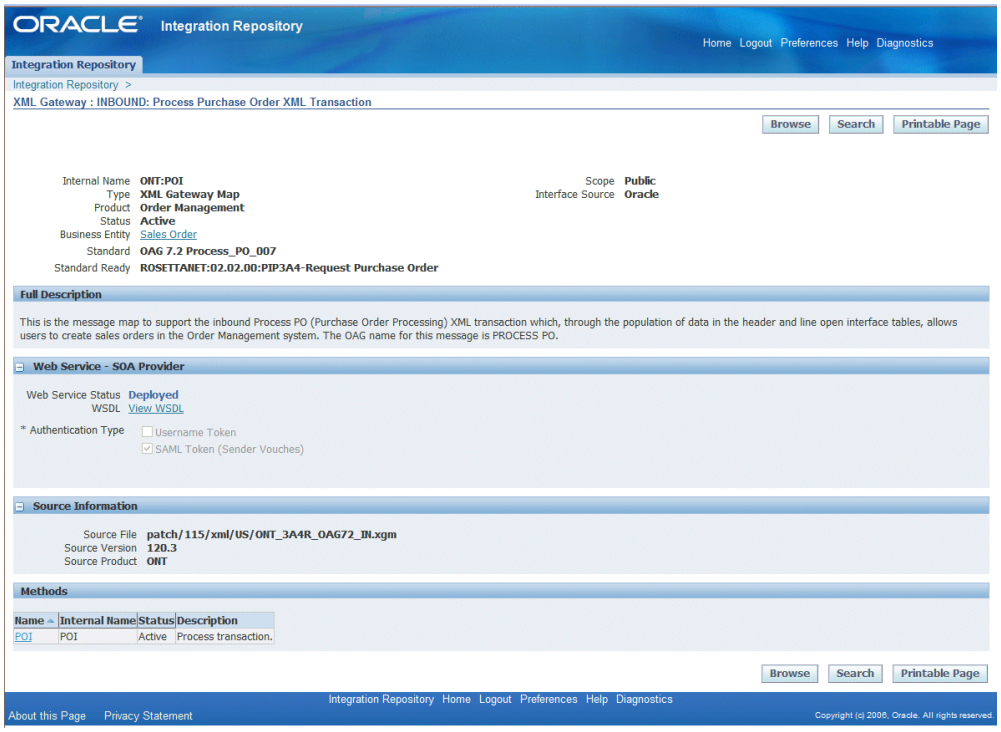
   For information on how to deploy a web service, see Deploying, Undeploying, and Redeploying Web Services, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

   Searching and Recording WSDL URL

   An integration developer also needs to log on to the system to locate and record the deployed web service WSDL URL for the inbound message map.

   This WSDL information will be used later in creating a partner link for the inbound map exposed as a web service during the BPEL process creation at design time.

XML Gateway Interface Details Page to Record a Deployed WSDL URL



For information on how to search for an interface and review the interface details, see Searching and Viewing Integration Interfaces, page 2-2.

Populating XML Gateway Header Variables

You need to populate certain variables in the BPEL PM in order to provide XML Gateway header information for Oracle E-Business Suite. The `MESSAGE_TYPE`, `MESSAGE_STANDARD`, `TRANSACTION_TYPE`, `TRANSACTION_SUBTYPE`, `DOCUMENT_NUMBER` and `PARTY_SITE_ID` are the mandatory header variables that you need to populate for the XML transaction to complete successfully.

Refer to Adding an Assign activity, page 5-19 for more information.

Ensuring Agent Listeners Are Up and Running

You need to ensure that listeners on the ECX_INBOUND, ECX_TRANSACTION queues are up and running. Use the following steps to configure these listeners in Oracle E-Business Suite:

1. Log in to Oracle E-Business Suite with the Workflow Administrator responsibility.
2. Click the **Workflow Administrator Web Applications** link from the Navigator.
3. Click the **Workflow Manager** link under Oracle Applications Manager.

4. Click the status icon next to **Agent Listeners**.

5. Configure and schedule the **ECX Inbound Agent Listener** and the **ECX Transaction Agent Listener**. Select the listener, and select Start from the **Actions** box. Click **Go** if they are not up and running.

BPEL Process Creation Flow

After deploying the BPEL process, you should get the same order information inserted into the Order Management table once a purchase order is created.

Based on the XML Gateway Inbound Process PO XML Transaction business scenario, the following design-time tasks are discussed in this chapter:

1. Create a new BPEL project, page 5-7

   Use this step to create a new BPEL project called `XMLGatewayInbound.bpel` using an Synchronous BPEL Process template. This automatically creates two dummy activities - Receive and Reply - to receive input from a trading partner and to reply output of the BPEL process back to the request application.

2. Create a Partner Link, page 5-9

   Use this step to create a partner link to allow the inbound message to be inserted to the Oracle E-Business Suite.

3. Add a Partner Link for File Adapter, page 5-11

   Use this step to add a partner link for File Adapter in order to pick up an XML file received from the trading partner to get the XML message.

4. Add Invoke activities, page 5-16

   Use this step to add two Invoke activities in order to:

   1. To get the XML message details that is received from the Receive activity.

   2. To enqueue the purchase order information to the ECX_INBOUND queue.

5. Add Assign activities, page 5-19

   Use this step to create two Assign activities in order to:

   1. To pass XML message obtained from the first Invoke activity to the second Invoke activity.

   2. To pass ECX header variables to the second Invoke activity as input variables.

For general information and basic concept of a BPEL process, see Understanding BPEL Business Processes, page D-1 and *Oracle BPEL Process Manager Developer's Guide* for details.

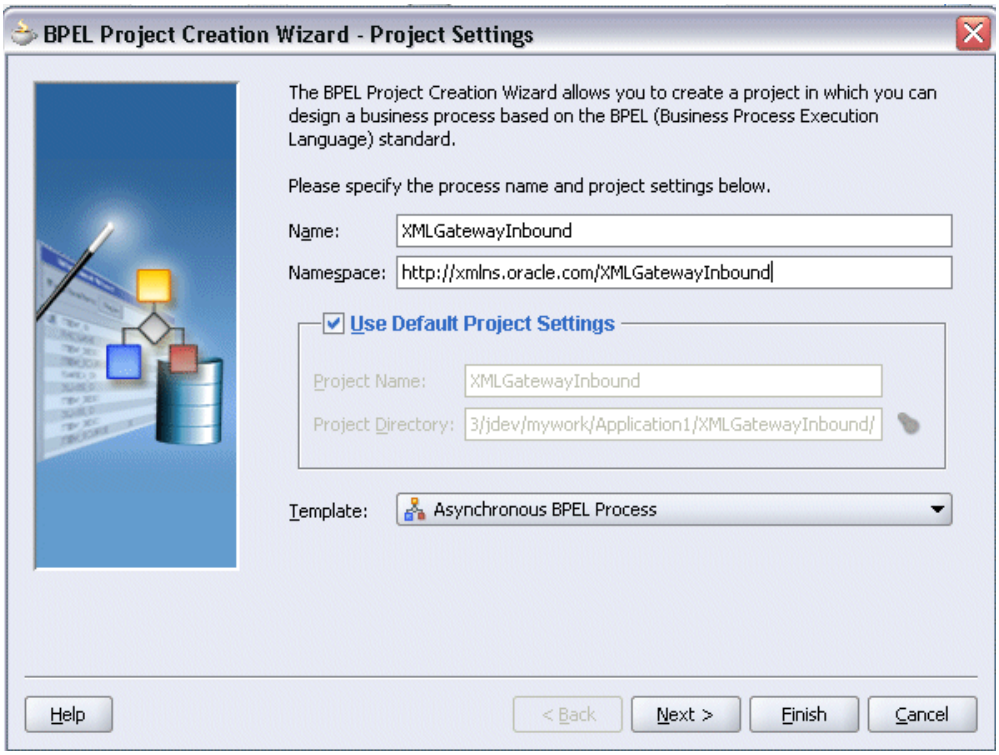
Creating a New BPEL Project

Use this step to create a new BPEL project that will contain various BPEL process activities.

To create a new BPEL project:

1. Open JDeveloper BPEL Designer.
2. From the **File** menu, select **New**. The New Gallery dialog box appears.
3. Select **All Items** from the **Filter By** box. This produces a list of available categories.
4. Expand the **General** node and then select **Projects**.
5. Select **BPEL Process Project** from the **Items** group.
6. Click **OK**. The BPEL Project Creation dialog box appears.

BPEL Project Creation Wizard - Project Settings Dialog



7. In the **Name** field, enter a descriptive name such as `XMLGatewayInbound`.

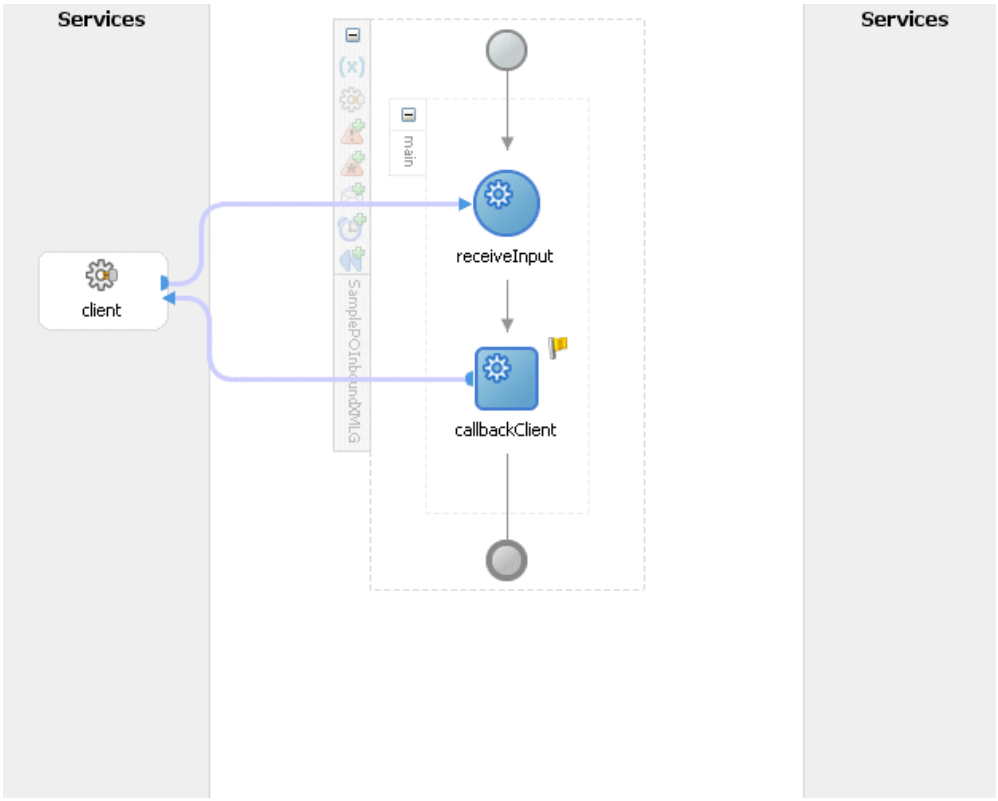
8. From the Template list, select **Asynchronous BPEL Process** and then select **Use Default Project Settings**.

9. Use the default input and output schema elements in the Input/Output Elements dialog box.

10. Click **Finish**.

    A new asynchronous BPEL process is created with the Receive and Callback activities. The required source files including bpel.xml, using the name you specified (for example, `XMLGInbound.bpel`) are also generated.

BPEL Process Diagram



Creating a Partner Link

Use this step to create a Partner Link called `ONT_POI` to insert sales order data to Oracle E-Business Suite.

To create a partner link to insert sales data to Oracle E-Business Suite:

1. In JDeveloper BPEL Designer, drag and drop the **PartnerLink** service from the Component Palette into the Partner Link border area of the process diagram. The Service Name dialog box appears.

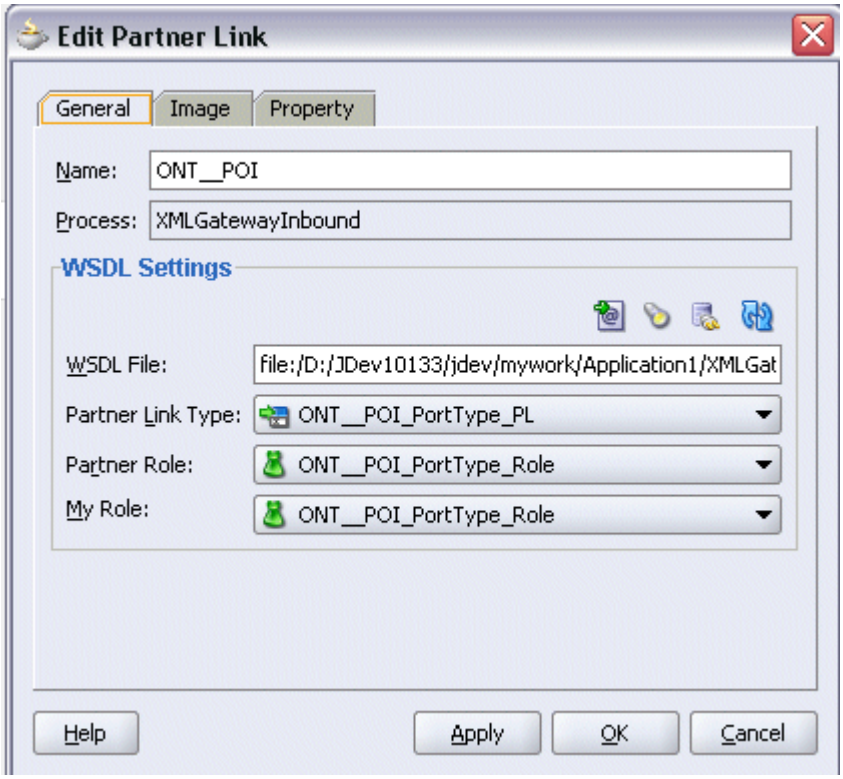
2. Copy the WSDL URL corresponding to the XML Gateway inbound map `INBOUND: Process Purchase Order XML Transaction (ONT:POI)` that you recorded earlier in the WSDL File field.

   A Partner Link Type message dialog box appears asking whether you want the system to create a new WSDL file that will by default create partner link types for you.

   Click **Yes** to have the Partner Name value populated automatically. The name is defaulted to ONT_POI.

Select Partner Role and My Role fields from the drop-down lists.

Edit Partner Link Dialog



Click **Apply**.

3. Select the Property tab and click the **Create Property** icon to select the following properties from the property name drop-down list in order to pass the security header along with the SOAP request:

   - wsseUsername

     Specify the username to be passed in the Property Value box.

   - wssePassword

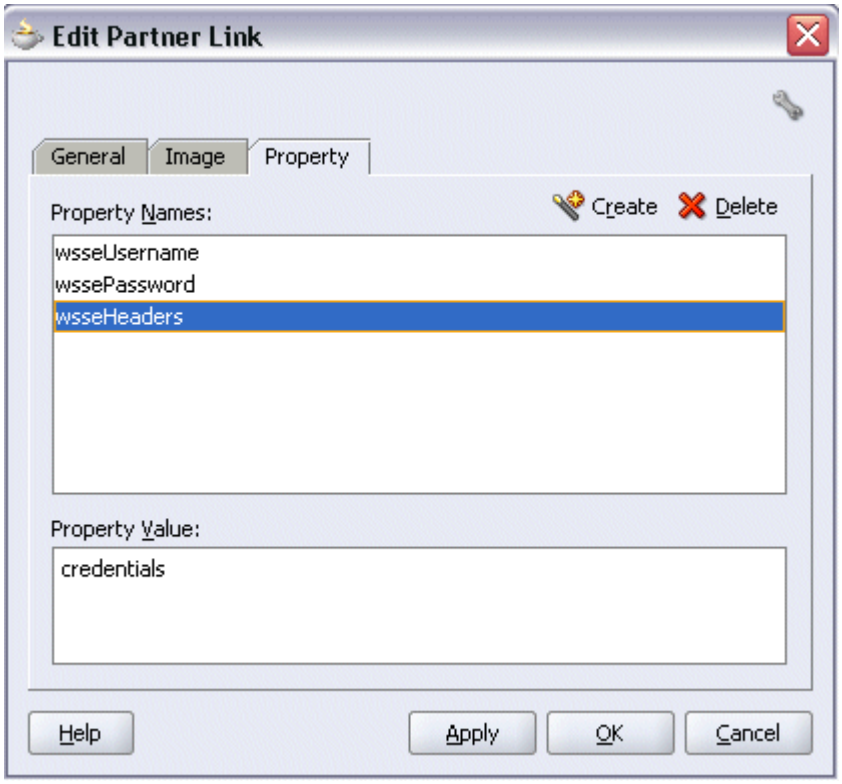
     Specify the corresponding password for the username to be passed in the Property Value box.

   - wsseHeaders

     Enter `credentials` as the property value.

   Click **Apply** to save the selected property values.

Edit Partner Link Dialog with Property Tab



4. Click **OK** to complete the partner link configuration. The partner link is created with the required WSDL settings, and is represented in the BPEL project by a new icon in the border area of the process diagram.

Adding Partner Links for File Adapter

Use this step to configure a BPEL process by adding a partner link for File Adapter to get the XML Message.

To add the Partner Link for File Adapter to get the XML Message:

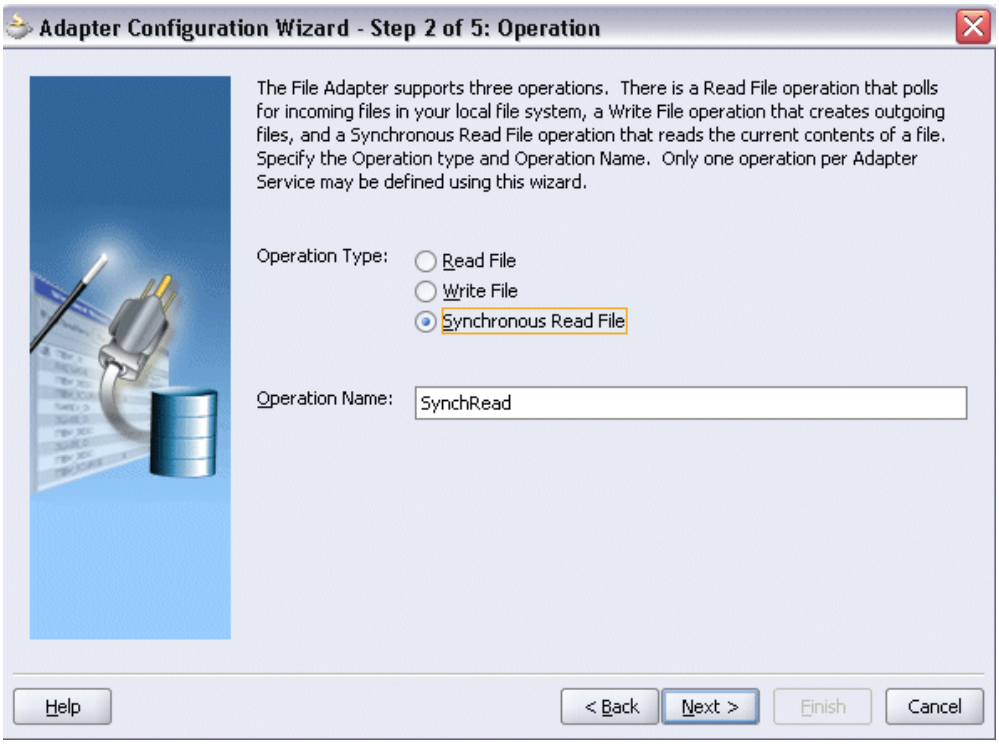
1. In JDeveloper BPEL Designer, drag and drop the **File Adapter** service from the **Adapter Service** section of the Component Palette into the Partner Link area of the process diagram. The Adapter Configuration wizard welcome page appears.

2. Click **Next**. The Service Name dialog box appears.

3. Enter a name for the file adapter service, such as `GetXMLMsg`. You can add an optional description of the service.

4. Click **Next** and the Operation dialog box appears.

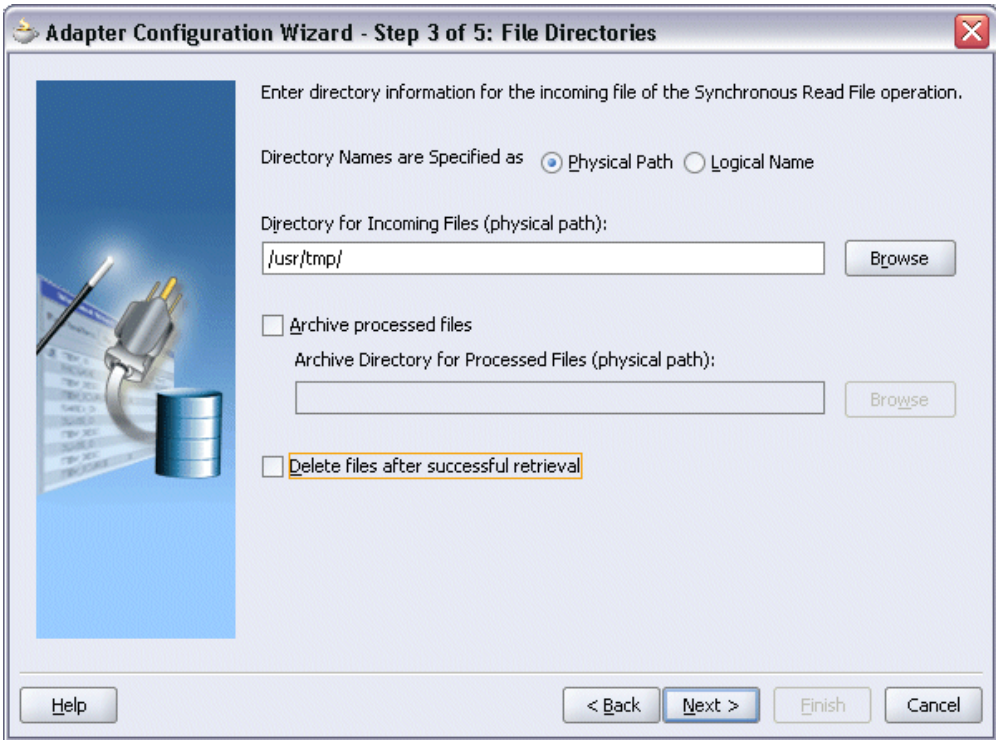
Operation Dialog



5. Specify the operation type, for example **Synchronous Read File**. This automatically populates the **Operation Name** field.

   Click **Next** to access the File Directories dialog box.

File Directories Dialog



6. Select **Physical Path** radio button and enter the physical path for incoming file directory information. For example, enter `/usr/tmp/`.

   > **Note:** To be able to locate the file from the physical directory you specified here, you must first place the input payload file (such as `order_data_xmlg.xml`) to the specified directory.

   Alternatively, click **Browse** to locate the incoming file directory information.

   Uncheck the **Delete files after successful retrieval** check box. Click **Next**.

7. Enter the name of the file for the synchronous read file operation. For example, enter `order_data_xmlg.xml`. Click **Next**. The Messages dialog box appears.

8. Select **Browse** to open the Type Chooser.

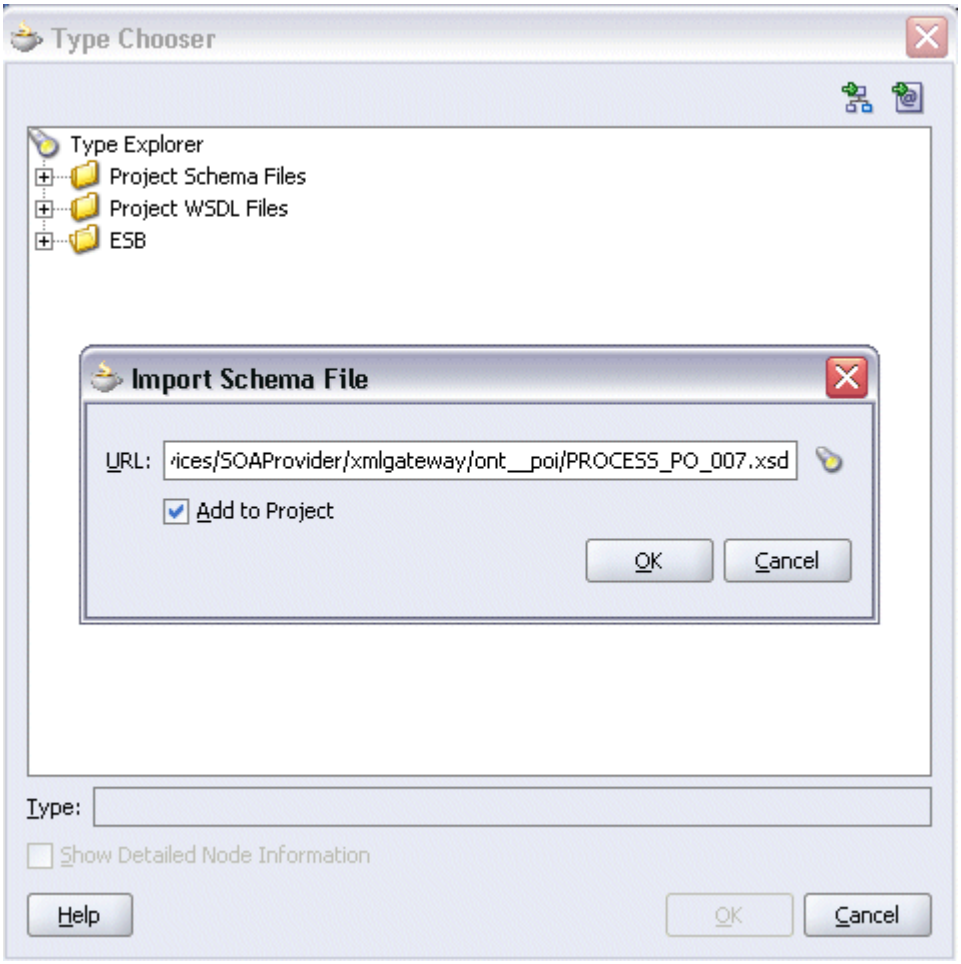
   Click **Import Schema Files** button on the top right corner of the Type Chooser window. This opens the Import Schema File pop-up window.

   Enter the schema location for the service. Such as `http://<myhost>:<port>/webservices/SOAProvider/xmlgateway/ont__poi/PROCESS_PO_007.xsd`.

   Schema location for your service can be found from the service WSDL URL (for

example, `http://<myhost>:<port>/webservices/SOAProvider/xmlgateway/ont__poi/?wsdl`).

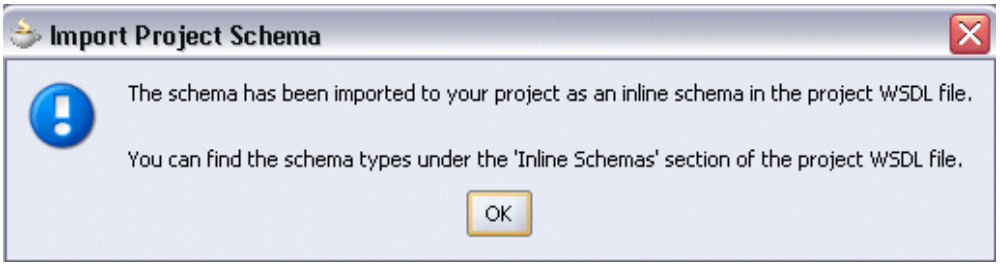
Import Schema File Pop-up Window



Select the **Add to Project** check box and click **OK**.

9. Click **OK** to the Import Project Schema message prompt.

Import Project Schema Message Prompt



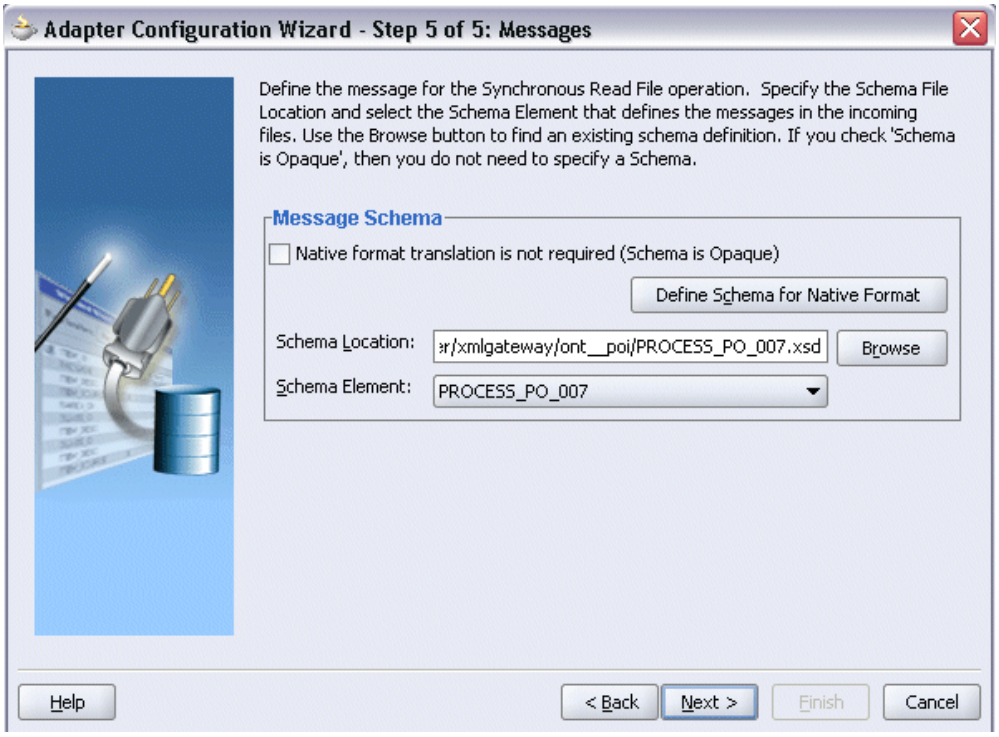
The Imported Schemas folder is automatically added to the Type Chooser window.

10. Expand the Imported Schemas folder and select **PROCESS_PO_007.xsd > PROCESS_PO_007**. Click **OK**.

    The selected xsd is displayed as Schema Location, and PROCESS_PO_007 is selected as Schema Element.

11. Click **OK** to populate the selected values in the Messages dialog box.

Messages Dialog with Selected Schema and Element



12. Click **Next** and then **Finish**. The wizard generates the WSDL file corresponding to the partner link. The main Create Partner Link dialog box appears, specifying the new WSDL file `GetXMLMsg.wsdl`.

    Click **Apply** and then **OK** to complete the configuration and create the partner link with the required WSDL settings for the File Adapter Service.

    The `GetXMLMsg` Partner Link appears in the BPEL process diagram.

Adding Invoke Activities

This step is to configure three Invoke activities:

1. To get the XML message details that is received from the Receive activity by invoking the `GetXMLMsg` partner link in an XML file.

2. To enqueue the purchase order information to the ECX_INBOUND queue by invoking `ONT_POI` partner link in an XML file.

To add the first Invoke activity for a partner link to get XML message:

1. In JDeveloper BPEL Designer, drag and drop the first **Invoke** activity from the Component Palette into the Activity box of the process diagram, between the **Receive** and **Callback** activities.

2. Link the Invoke activity to the `GetXMLMsg` service. The Edit Invoke dialog box appears.

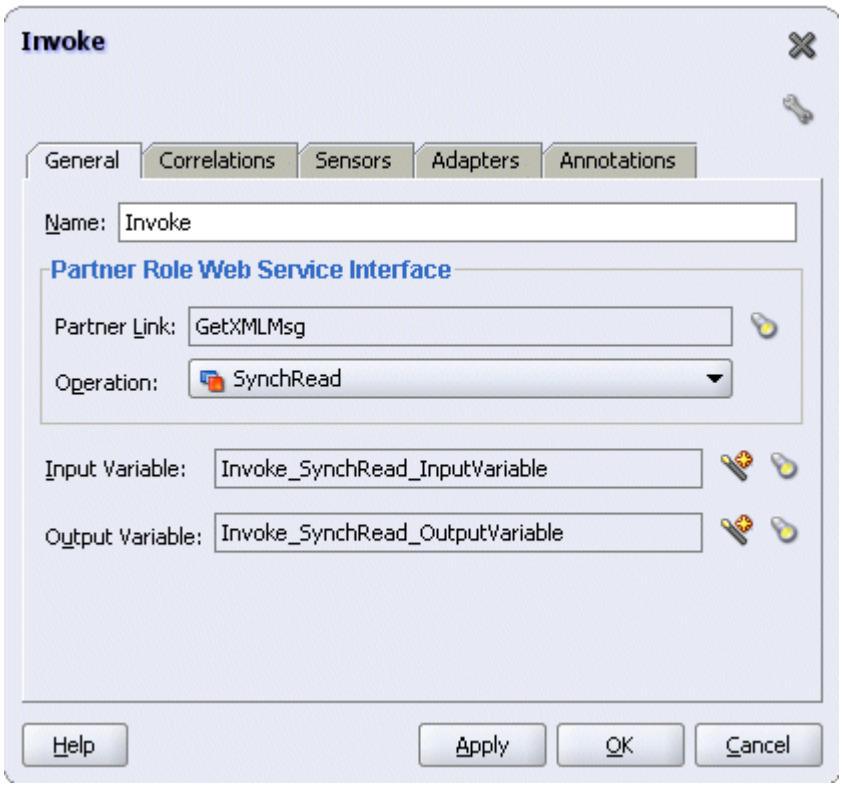
3. Enter a name for the Invoke activity and then click the **Create** icon next to the **Input Variable** field to create a new variable. The Create Variable dialog box appears.

4. Select **Global Variable** and then enter a name for the variable. You can also accept the default name. Click **OK**.

5. Enter a name for the Invoke activity and click the **Create** icon next to the **Output Variable** field to create a new variable. The Create Variable dialog box appears.

6. Select **Global Variable**, and enter a name for the variable. You can also accept the default name. Click **OK**.

Edit Invoke Dialog



Click **Apply** and then **OK** in the Edit Invoke dialog box to finish configuring the Invoke activity.

The first Invoke activity appears in the process diagram.

To add the second Invoke activity for a partner link to enqueue PO information:

1. In JDeveloper BPEL Designer, drag and drop the second **Invoke** activity from the

Component Palette into the Activity box of the process diagram, between the first **Invoke** and **Callback** activities.

2. Link the Invoke activity to the `ONT_POI` service. The Edit Invoke dialog box appears.

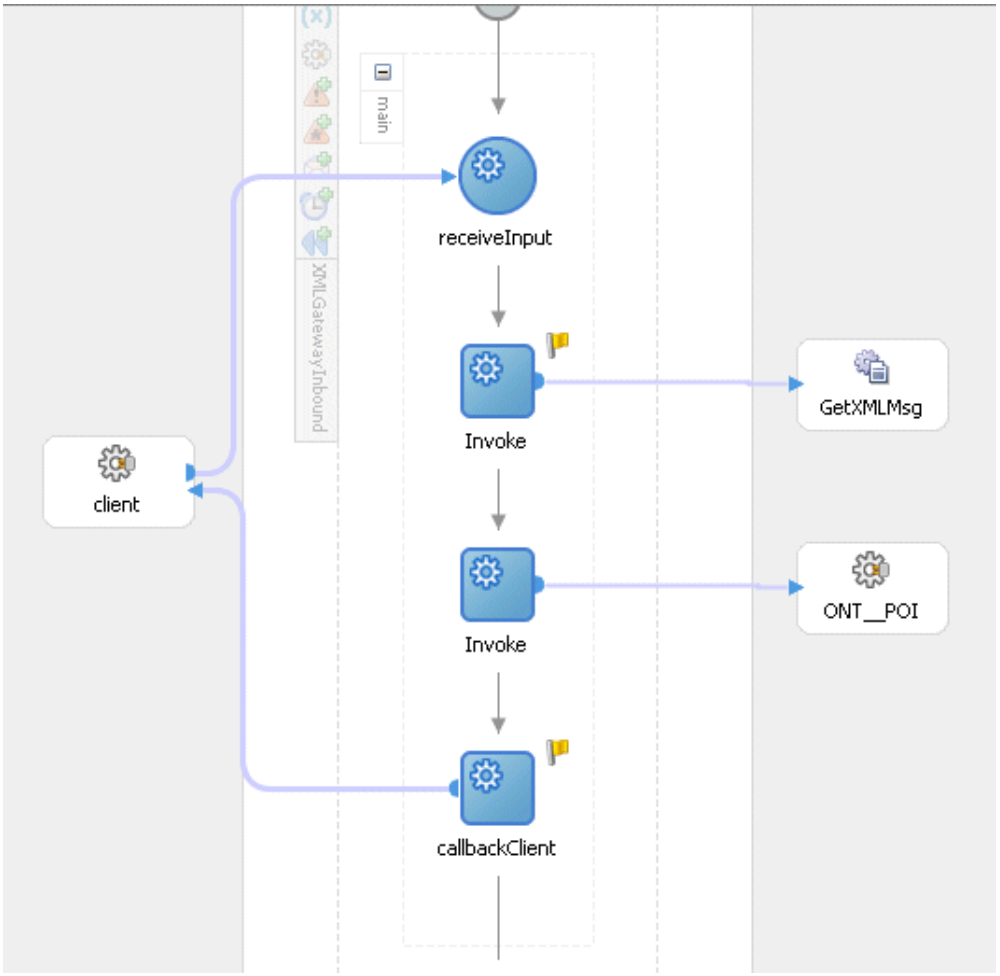
3. Enter a name for the Invoke activity and then click the **Create** icon next to the **Input Variable** field to create a new variable. The Create Variable dialog box appears.

4. Select **Global Variable** and then enter a name for the variable. You can also accept the default name. Click **OK**.

   Click **Apply** and then **OK** in the Edit Invoke dialog box to finish configuring the Invoke activity.

5. The process diagram appears.

BPEL Process Diagram with Two Invoke Activities



Adding Assign Activities

This step is to configure two Assign activities:

1. To pass XML message as an input to the Invoke activity for enqueuing message.
2. To pass XML Gateway header variables as input variables to the Invoke activity in order to provide context information for Oracle E-Business Suite.

To add the first Assign activity to pass XML message as input to the Invoke activity:

1. In JDeveloper BPEL Designer, drag and drop the **Assign** activity from the Component Palette into the Activity box of the process diagram, between the two **Invoke** activities.
2. Double-click the **Assign** activity to access the Edit Assign dialog box.

3. Click the General tab to enter the name for the Assign activity, such as 'SetOrderXML'.

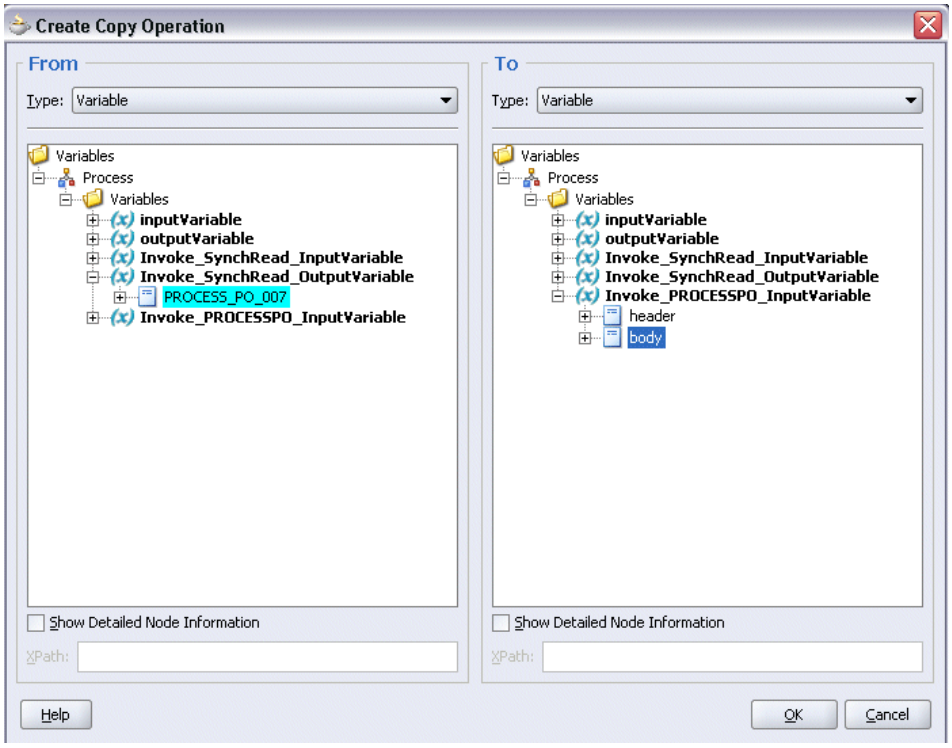
4. On the Copy Operation tab, click **Create** and then select **Copy Operation** from the menu. The Create Copy Operation window appears.

5. Enter the following information:

   - In the From navigation tree, select type Variable. Navigate to **Variable > Process > Variables > Invoke_SynchRead_OutputVariable** and select **Process_PO_007**.

   - In the To navigation tree, select type Variable. Navigate to **Variable > Process > Variables > Invoke_PROCESSPO_InputVariable** and select **body**.

Create Copy Operation Dialog to Assign Parameters



   - Click **OK**. The Edit Assign dialog box appears.

6. Click **Apply** and then **OK** to complete the configuration of the Assign activity.

To add the second Assign activity to pass XML Gateway header variables to the Invoke activity:

1. Add the second Assign activity by dragging and dropping the **Assign** activity from

the Component Palette into the Activity box of the process diagram, between the `SetOrderXML` **Assign** activity and the second **Invoke** activity.

2. Repeat Step 2 to Step 4 described in creating the first Assign activity to add the second Assign activity called 'SetECXHeader'.

3. On the Copy Operation tab, click **Create** and then select **Copy Operation** from the menu. The Create Copy Operation window appears.

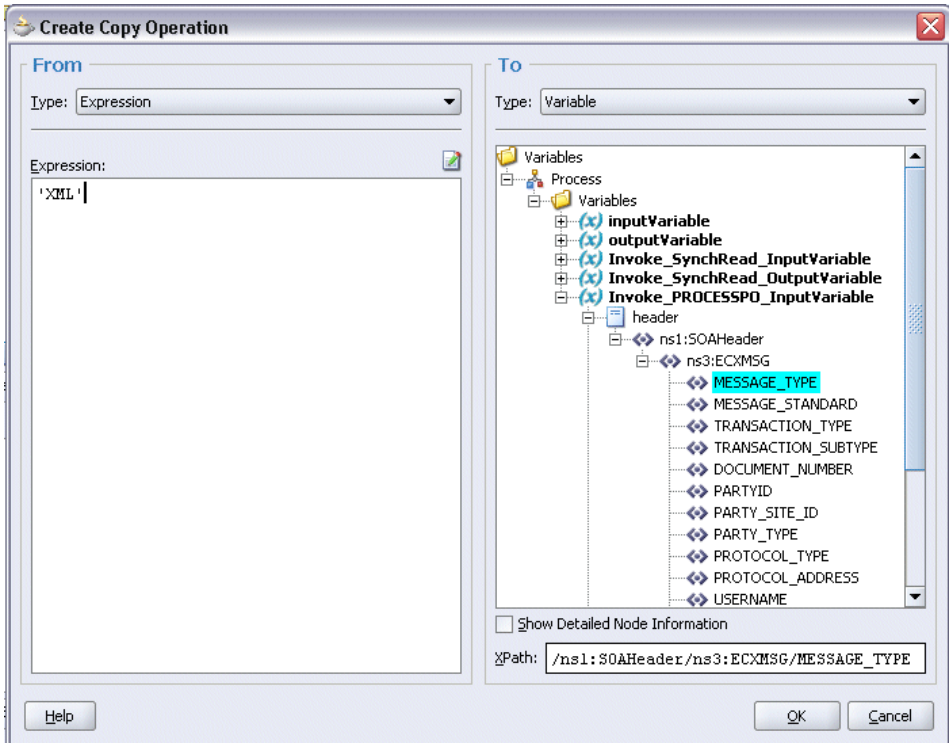
4. Enter the first pair of parameters:

   - In the From navigation tree, select type Expression and then enter 'XML' in the Expression box.

   - In the To navigation tree, select type Variable. Navigate to **Variable > Process > Variables > Invoke_PROCESSPO_InputVariable > header > ns1:SOAHeader > ns3:ECXMSG** and select **MESSAGE_TYPE**.

     The XPath field should contain your selected entry.

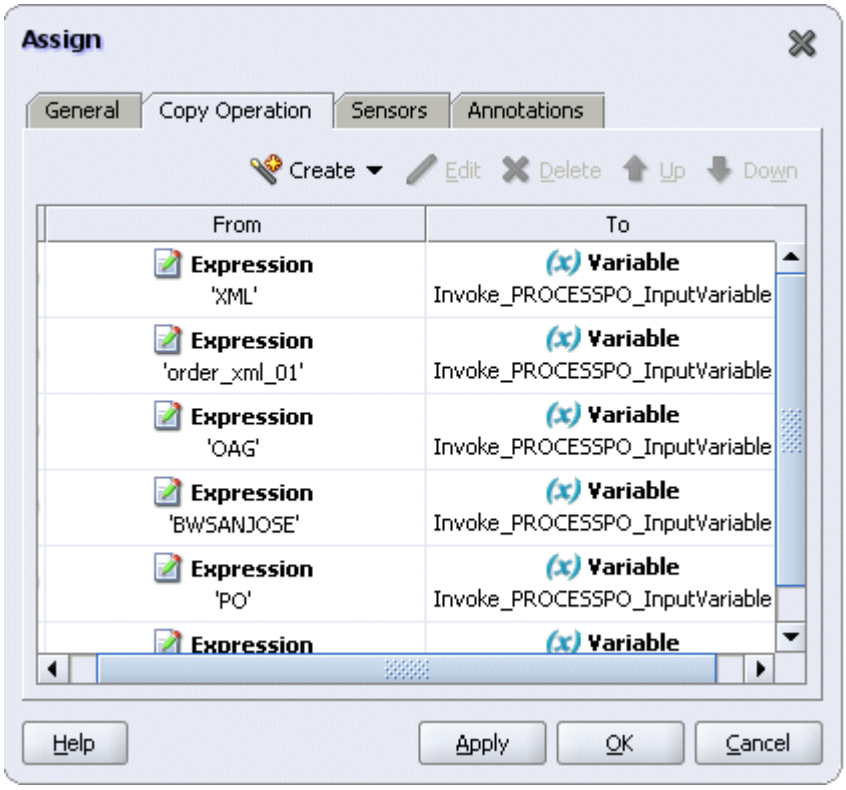
Create Copy Operation Dialog to Assign Value in the Expression Region



- Click **OK**.

5. Use the same mechanism described in step 3 and 4 to enter the following additional parameters:
   - MESSAGE_STANDARD: 'OAG'
   - TRANSACTION_TYPE: 'PO'
   - TRANSACTION_SUBTYPE: 'PROCESS'
   - DOCUMENT_NUMBER: 'order_xml_01'
   - PARTY_SIDE_ID: 'BWSANJOSE'

Edit Assign Dialog with Copy Operation Tab



6. Click **Apply** and **OK** to complete the configuration of the Assign activity.

Deploying and Testing the BPEL Process at Run Time

After creating a BPEL process using the WSDL URL generated from the XML Gateway inbound message map interface definition, you can deploy it to a BPEL server if needed. To ensure that this process is modified or orchestrated appropriately, you can also manually test the BPEL process by initiating the business process contained in the BPEL process to test the interface integration.

Prerequisites

Before deploying the BPEL process using Oracle JDeveloper, you must ensure that you have established the connectivity between the deign-time environment and the run-time servers including the application server and the integration server.

How to configure the necessary server connection, see Configuring Server Connection, page B-1.

To validate your BPEL process, perform the following run-time tasks:

1. Deploy the BPEL process, page 5-24

Once you deploy the process to a BPEL server, it becomes available so that you can run the process manually to test it for validation.

2. Test the BPEL process, page 5-25

   After deploying a BPEL process, you can manage the process from the BPEL console to validate the interface integration contained in your BPEL process.

Deploying the BPEL Process

You must deploy the BPEL process (`XMLGatewayInbound.bpel`) that you created earlier before you can run it. The BPEL process is first compiled and then deployed to the BPEL server.

> **Note:** Before deploying the BPEL Process for XML Gateway Inbound service, you should:
>
> - Load the `order_data_xmlg.xml` file into the specified directory '/usr/tmp/' folder of SOA Suite server (or D:\HOL in case of SOA Server in Windows machine).
>
> - Edit the input file `order_data_xmlg.xml` by entering values for <REFERENCEID> and <POID> such as 'order_xml_01'.

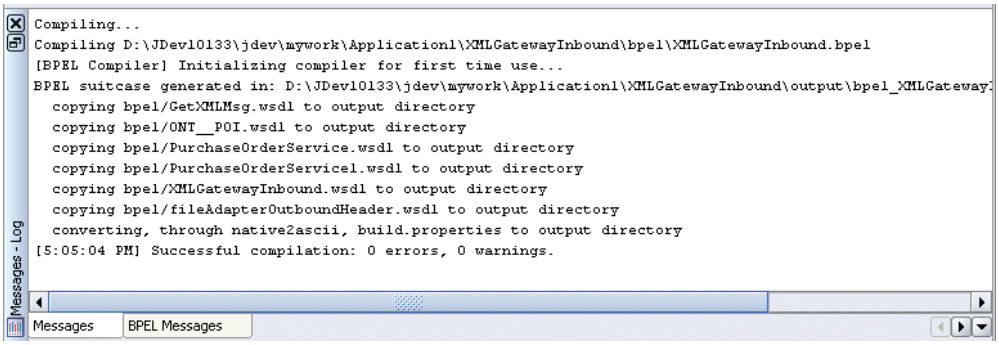
To deploy the BPEL process:

1. In the Applications Navigator of JDeveloper BPEL Designer, select the **XMLGInbound** project.

2. Right-click the project and click **Make** action from the menu.

   Look for any compilation error messages in Message Log.

Messages Window



Right-click the project and select **Deploy >*Integration Server Connection name*> Deploy to Default Domain** action from the menu.

For example, you can select **Deploy > BPELServerConn > Deploy to Default Domain** to deploy the process if you have the BPEL Process Manager setup appropriately.

3. Look for 'Build successful' message in Apache Ant – Log to ensure that the BPEL project is compiled and successfully deployed.

Testing the BPEL Process

Once the BPEL process is deployed, it can be seen in the BPEL console. You can manage and monitor the process from the BPEL console. You can also test the process and the integration interface by manually initiating the process.

To test the BPEL process:

1. Log in to Oracle Application Server 10g BPEL Console (`http://<soaSuiteServerHostName>:<port>/BPELConsole`). The BPEL Console login page appears.

2. Enter the username and password and click **Login**.

   The Oracle Enterprise Manager 10g BPEL Control appears.

3. In the BPEL Console, confirm that XMLGInbound has been deployed.

4. Click the XMLGInbound link to open the Initiate tab

5. Click **Post XML Message** to initiate the process.

Verifying Records in Oracle E-Business Suite

Once the BPEL process is successfully initiated and completed, you can validate it through the relevant module in Oracle E-Business Suite.

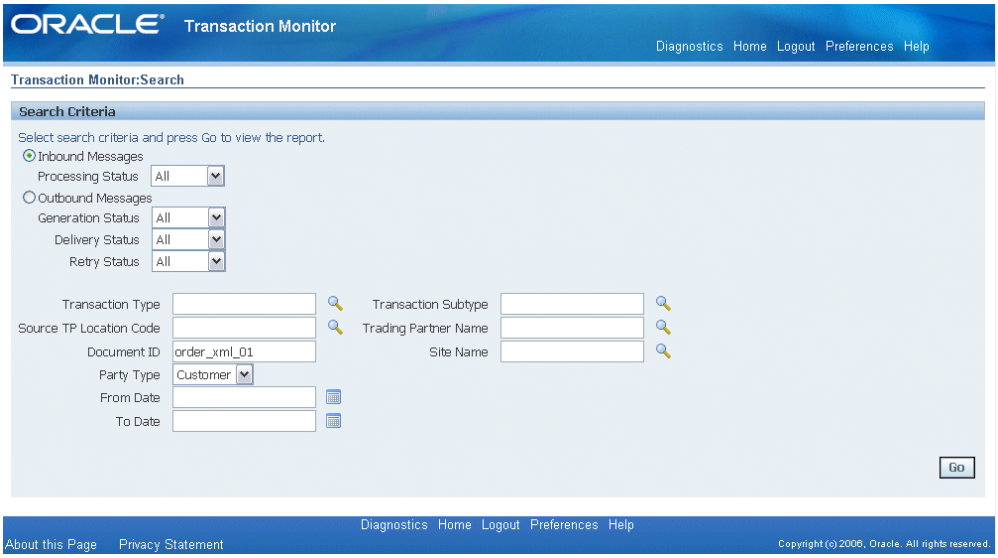
To validate it in Oracle Transaction Monitor:

You can validate it from the Transaction Monitor. The Transaction Monitor is a tool for monitoring the status of inbound and outbound transactions originating from and going into Oracle E-Business Suite that have been processed by the XML Gateway and delivered or received by the Oracle Transport Agent. It shows a complete history and audit trail of these documents.

1. Log in to Oracle E-Business Suite with the Workflow Administrator Web Applications responsibility.

   Select the Transaction Monitor link to open the search window to search for the order.

Transaction Monitor: Search Page



2. Clear From Date and To Date fields and enter 'order_xml_01' in the Document ID field.

3. Select Customer as the Party Type. Click **Go** to execute the search.

   This retrieves XML inbound transaction 'order_xml_01' in the Inbound Search Results region.

4. Confirm that the transaction 'order_xml_01' has status 'SUCCESS'.

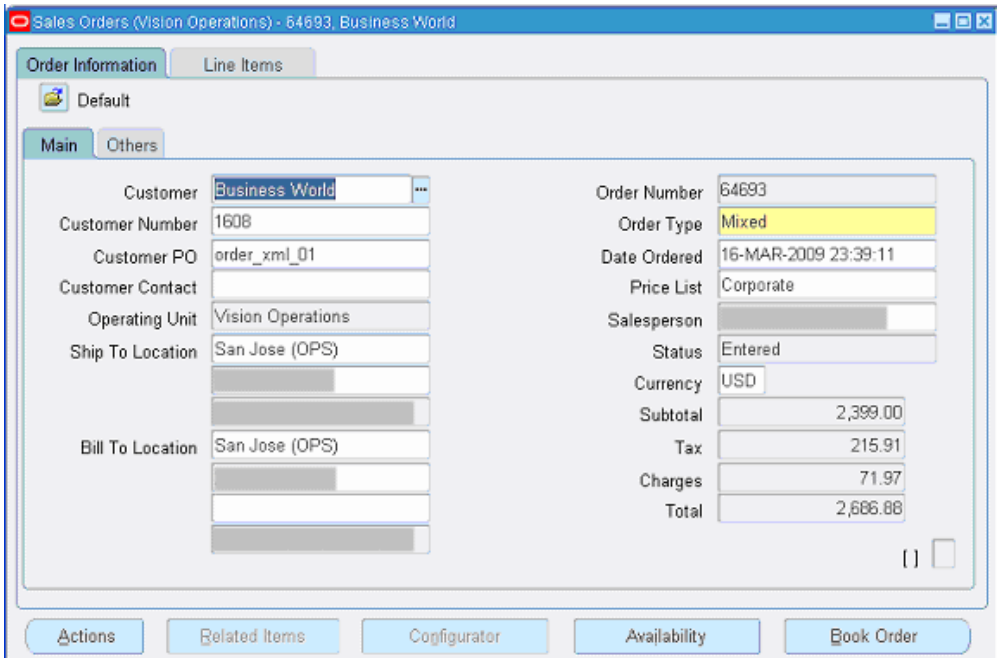
To validate it in Oracle Order Management:

1. Log in to the Forms-based Oracle E-Business Suite with the Order Management, Super User responsibility.

2. Select **Order Returns > Sales Order**. Sales Order Forms would open up.

3. Search for an order by entering the order number in the Customer PO field (such as 'order_xml_01'). This would bring up the details of newly created order.

Sales Orders Forms



You can also select the Items tab for item details.

Using XML Gateway Outbound Through Subscription Model

This section includes the following topics:

- Using XML Gateway Outbound Messages in Creating a BPEL Process at Design Time, page 5-27
- Deploying and Testing a BPEL Process at Run Time, page 5-41

Using XML Gateway Outbound Services at Design Time

For an outbound XML Gateway Map interface, since an outbound message is first enqueued to the ECX_OUTBOUND queue, Oracle E-Business Suite Integrated SOA Gateway supports it through subscription model by first dequeuing the message to retrieve outbound data and then invoking an appropriate outbound XML Gateway map to update Oracle E-Business Suite.

BPEL Process Scenario

Take XML Gateway outbound interface 'PO acknowledgement XML Transaction' as an example. The XML Gateway outbound interface is exposed as a Web service through `ECX_CBODO_OAG62_OUT` outbound map.

When a purchase order is created and approved, on approval of the purchase order, a workflow will be triggered which creates the Purchase Order Acknowledgement flow and sends out the PO Acknowledgement as an XML file. The workflow delivers the Confirm BOD as the PO Acknowledgement to ECX_OUTBOUND queue for delivery to the other system.

The correlation Id for this message is set to "BPEL" and the Oracle BPEL PM listens to ECX_OUTBOUND queue for the message with the correlation Id = "BPEL". Confirm BOD as the PO Acknowledgement is written as an output XML file using File Adapter.

If the BPEL process is successfully executed after deployment, you should get the same order book reference ID (Customer PO) information from the output XML file once a purchase order is approved.

Prerequisites to Create a BPEL Process Using XML Gateway Outbound Messaging

You need to set up the correlation identifier in Oracle E-Business Suite. The correlation identifier enables you to label messages meant for a specific agent, in case there are multiple agents listening on the outbound queue. The agent listening for a particular correlation picks up the messages that match the correlation identifier for the agent.

To set up the correlation identifier:

1. Log in to Oracle E-Business Suite with the XML Gateway responsibility. The Navigator page appears.
2. Click the **XML Gateway** link.
3. Click the **Define Lookup Values** link under XML Gateway.
4. Search for `COMM_METHOD` in the **Type** field to see if it exists in the system.
5. Add a new record to the `COMM_METHOD` type by entering `BPEL` for the **Code** field and **Meaning** field. Enter description information and save the record.

   Oracle XML Gateway puts the correlation of BPEL when enqueueing the message on the ECX_OUTBOUND queue.

Once you have the correlation identifier set up correctly, you also need to ensure the trading partner that you want to use has the Protocol Type field set to `BPEL`.

BPEL Process Creation Flow

Based on the PO acknowledgement XML Transaction scenario, the following design-time tasks are discussed in this chapter:

1. Create a new BPEL project, page 5-29

   Use this step to create a new BPEL project called `XMLGOutbound.bpel`.

2. Create a Partner Link for AQ Adapter, page 5-30

   Use this step to dequeue the event details from the ECX_OUTBOUND queue.

3. Add a Receive activity, page 5-35

   Use the Receive activity to take PO acknowledgement details as an input to the Assign activity.

4. Add a Partner Link for File Adapter, page 5-36

   This is to write PO acknowledgement details in an XML file as an output file.

5. Add an Invoke activity, page 5-39

   This is to write PO acknowledgement information to an XML file through invoking the partner link for File Adapter.

6. Add an Assign activity, page 5-40

   Use the Assign activity to take the output from the Receive activity and to provide input to the Invoke activity.

For general information and basic concept of a BPEL process, see Understanding BPEL Business Processes, page D-1 and *Oracle BPEL Process Manager Developer's Guide* for details.

Creating a New BPEL Project

Use this step to create a new BPEL project that will contain various BPEL process activities.

To create a new BPEL project:

1. Open JDeveloper BPEL Designer.
2. From the **File** menu, select **New**. The New Gallery dialog box appears.
3. Select **All Technologies** from the **Filter By** box. This produces a list of available categories.
4. Expand the **General** node and then select **Projects**.
5. Select **BPEL Process Project** from the **Items** group.
6. Click **OK**. The BPEL Process Project dialog box appears.
7. In the **Name** field, enter a descriptive name such as `XMLGOutbound`.
8. From the Template list, select **Empty BPEL Process** and then select **Use Default Project Settings**.
9. Click **Finish**.

   A new BPEL project is created with the required source files including bpel.xml,

using the name you specified (for example, `XMLGOutbound.bpel`).

Creating a Partner Link for AQ Adapter

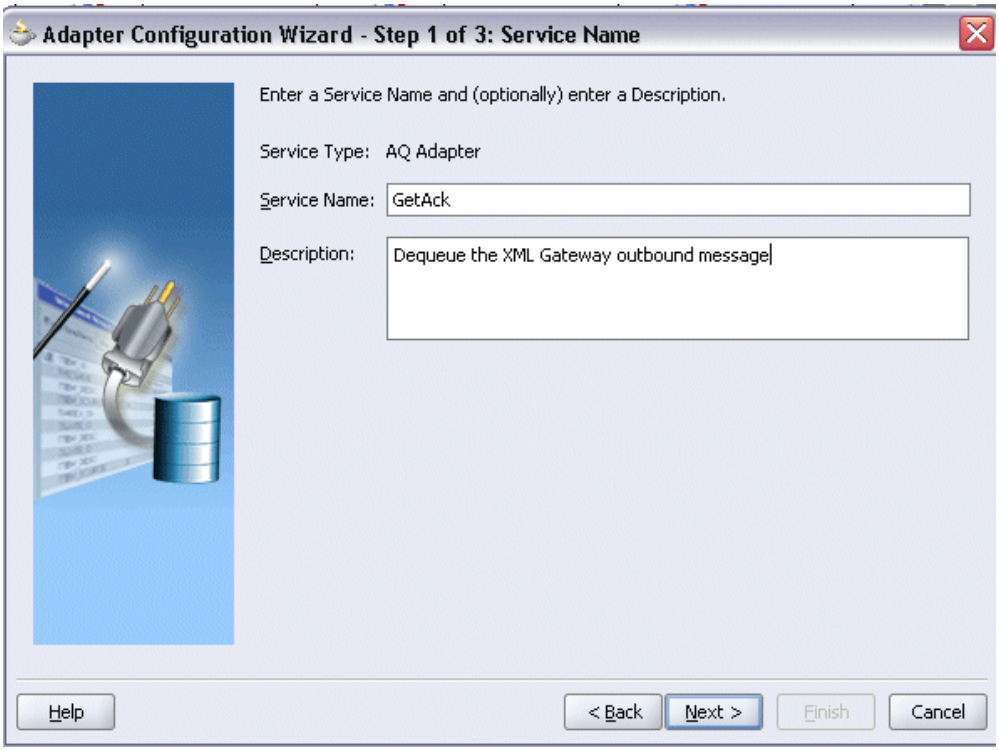
Use this step to create a Partner Link called `GetAck` for AQ Adapter to dequeue the XML Gateway outbound message (for example, `ECX_CBODO_OAG62_OUT`) in the `ECX_OUTBOUND` queue.

To create a partner link for AQ Adapter:

1. In JDeveloper BPEL Designer, drag and drop the **AQ Adapter** service from the Component Palette into the Partner Link border area of the process diagram. The Adapter Configuration Wizard appears.

2. Enter a service name in the Service Name dialog box, for example `GetAck`. You can also add an optional description of the service.

Service Name Dialog



3. Click **Next**. The Service Connection dialog box appears.

4. You can use an existing database connection by selecting a database connection from the **Connection** list or define a new database connection by clicking **New** to open the Create Database Connection Wizard.

Note: You need to connect to the database where Oracle E-Business Suite is running.

To create a new database connection:

1. Click **New** to open the Create Database Connection Wizard. Click **Next** and enter an unique connection name and then select a connection type, such as Oracle (JDBC), for the database connection. Click **Next**.

2. Enter an appropriate username and password to authenticate the database connection in the Authentication dialog box. Click **Next**

3. Specify the following information in the Connection dialog box:

   - Driver: Thin

   - Host Name: Enter the host name for the database connection. For example, `myhost01.example.com`.

   - JDBC Port: Enter JDBC port number (such as `1521`) for the database connection.

   - SID: Specify an unique SID value (such as `sid01`)for the database connection.

4. Click **Next** to test your database connection.

   The status message `"Success!"` indicates a valid connection.

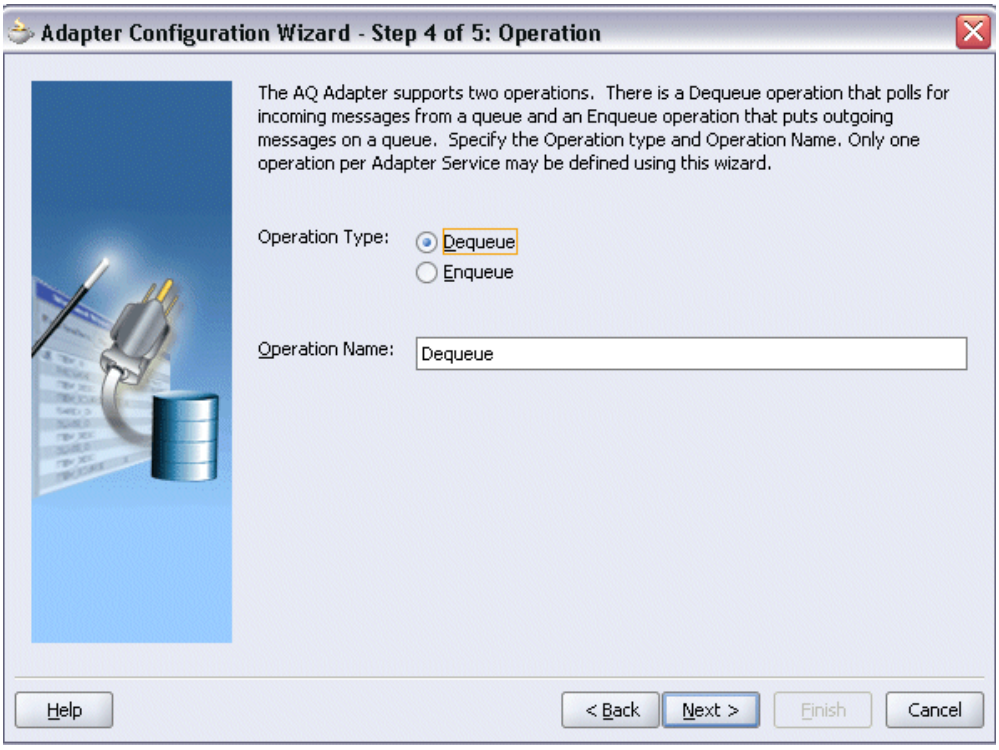
5. Click **Next** to return to the Service Connection dialog box providing a summary of the database connection.

5. The JNDI (Java Naming and Directory Interface) name corresponding to the database connection you specified appears automatically in the **JNDI Name** field of the Service Connection dialog box. Alternatively, you can enter a different JNDI name.

6. Click **Next** to open Operation dialog box.

   Select **Dequeue** radio button and this selected value is also populated in the Operation Name field.

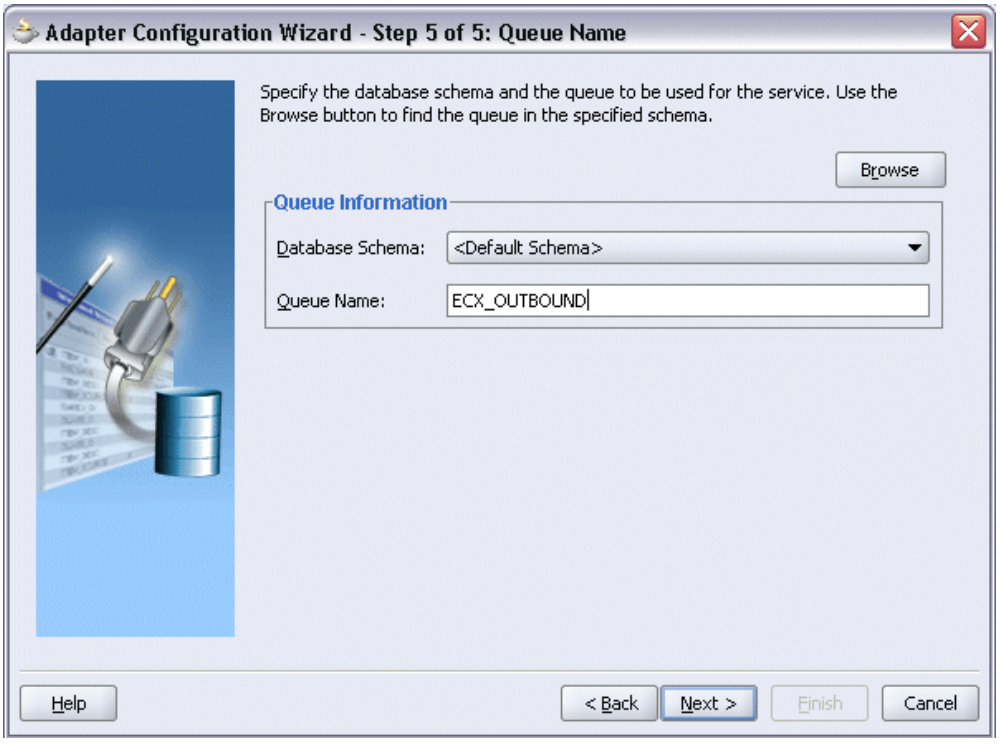
Operation Dialog



7. Click **Next** to open the Queue Name dialog box.

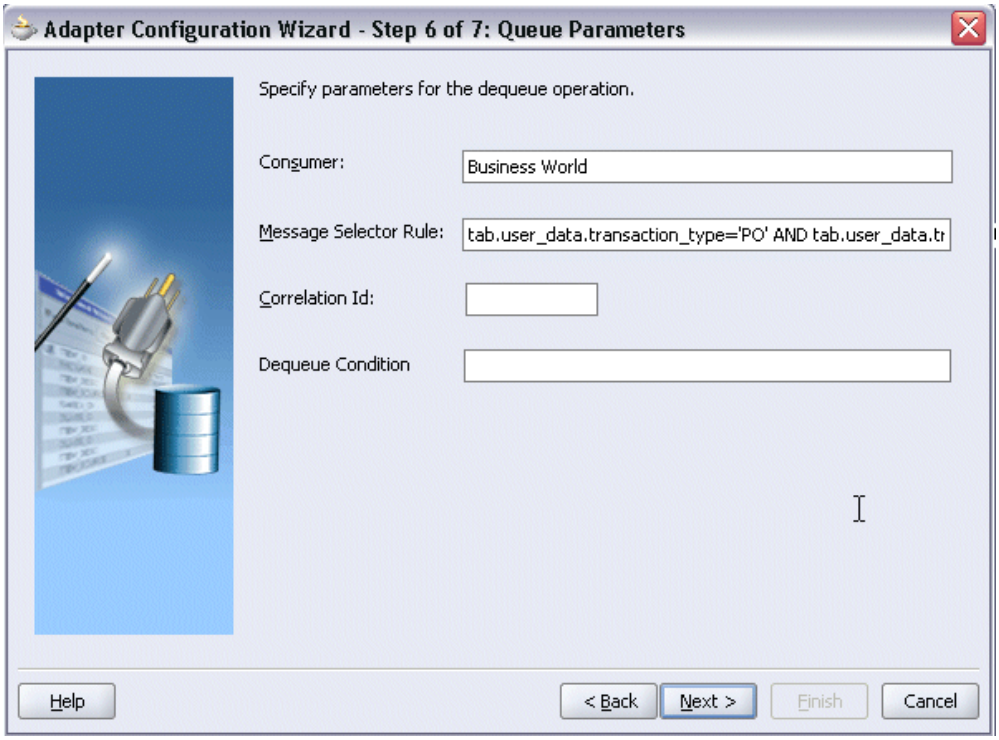
   Select Default Schema as the Database Schema field. Enter 'ECX_OUTBOUND' as the Queue Name field.

Queue Name Dialog



8. Click **Next** to open the Queue Parameter dialog box.

Queue Parameters Dialog



Enter the following information:

- Enter an unique consumer name.
- Enter message selector rule information (such as `tab.user_data.transaction_type='PO' AND tab.user_data.transaction_subtype='POO'`).

9. Click **Next**. The Messages dialog box opens.

   Click **Browse** to open the Type Chooser window to select CONFIRM_BOD_002.xsd as the Schema Location and CONFIRM_BOD_002 as the Schema Element.

10. Click **Next** to proceed to the Finish dialog box to confirm that you have finished defining the AQ Adapter for the `GetAck` service.

11. Click **Finish**. The wizard generates the WSDL file corresponding to the `GetAck` service.

    Click **Apply** and then **OK** to complete the partner link configuration. The partner link is created with the required WSDL settings, and is represented in the BPEL project by a new icon in the border area of the process diagram.

Adding a Receive Activity

This step is to configure a Receive activity to receive XML data from the partner link `GetAck` that you configured for the AQ adapter service.

The XML data received from the Receive activity is used as an input variable to the Assign activity that will be created in the next step.

To add a Receive activity:

1. In JDeveloper BPEL Designer, drag and drop the **Receive** activity from the **BPEL Activities** section of the Component Palette into the Activity box of the process diagram.

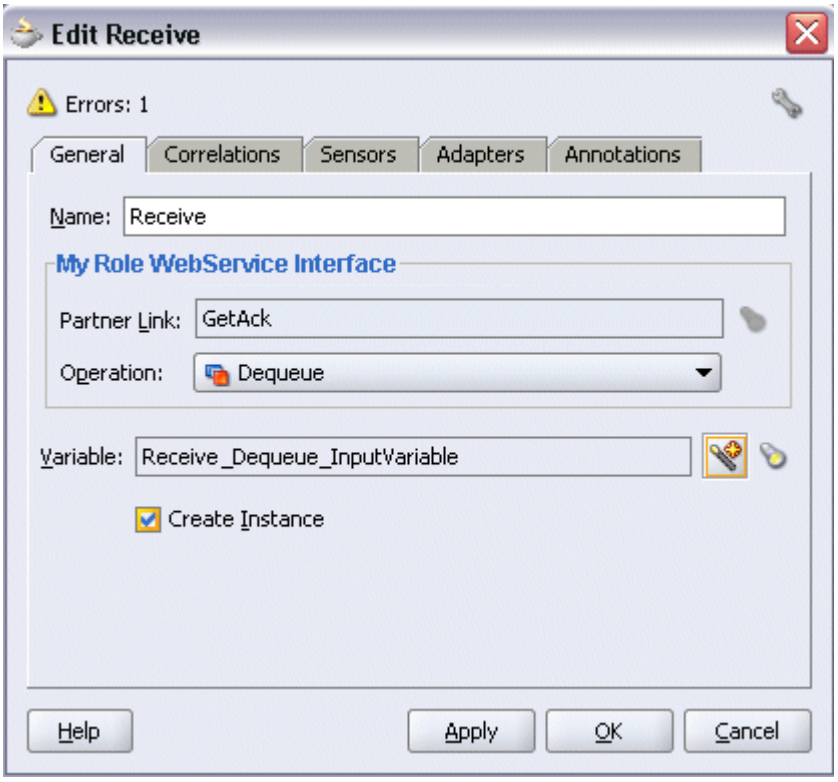
2. Link the Receive activity to the `GetAck` partner link. The Receive activity will take event data from the partner link. The Edit Receive dialog box appears.

3. Enter a name for the receive activity. Click the **Create** icon next to the **Variable** field to create a new variable. The Create Variable dialog box appears.

4. Select **Global Variable**, and then enter a name for the variable. You can accept the default name. Click **OK** to return to the Edit Receive dialog box.

5. Select **Create Instance** check box. Click **Apply** and **OK** to finish configuring the Receive activity.

Edit Receive Dialog



The Receive activity appears in the BPEL process diagram.

Adding a Partner Link for File Adapter

Use this step to configure a partner link by writing the purchase order acknowledgement to an XML file.

To add a Partner Link for File Adapter:

1. In JDeveloper BPEL Designer, drag and drop the **File Adapter** service from the **Adapter Service** section of the Component Palette into the Partner Link area of the process diagram. The Adapter Configuration wizard appears.

2. The Service Name dialog box appears.

3. Enter a name for the File Adapter service, such as `WriteAck`. You can add an optional description of the service.

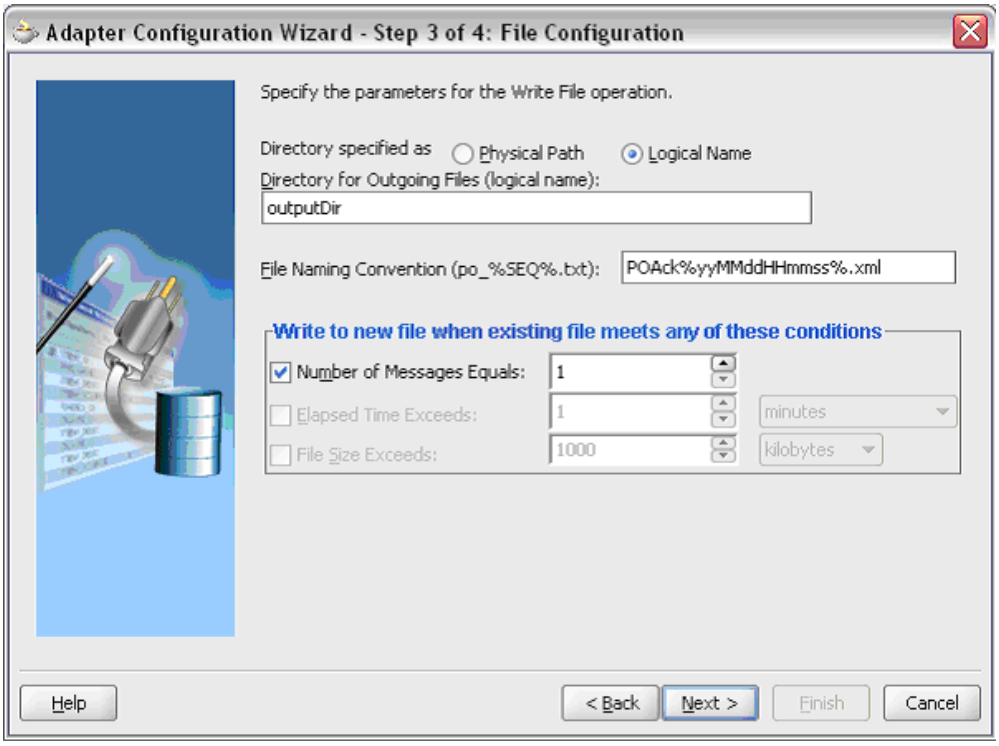
4. Click **Next** and the Operation dialog box appears.

Operation Dialog

5. Specify the operation type, for example **Write File**. This automatically populates the **Operation Name** field.

   Click **Next** to access the File Configuration dialog box.

File Configuration Dialog



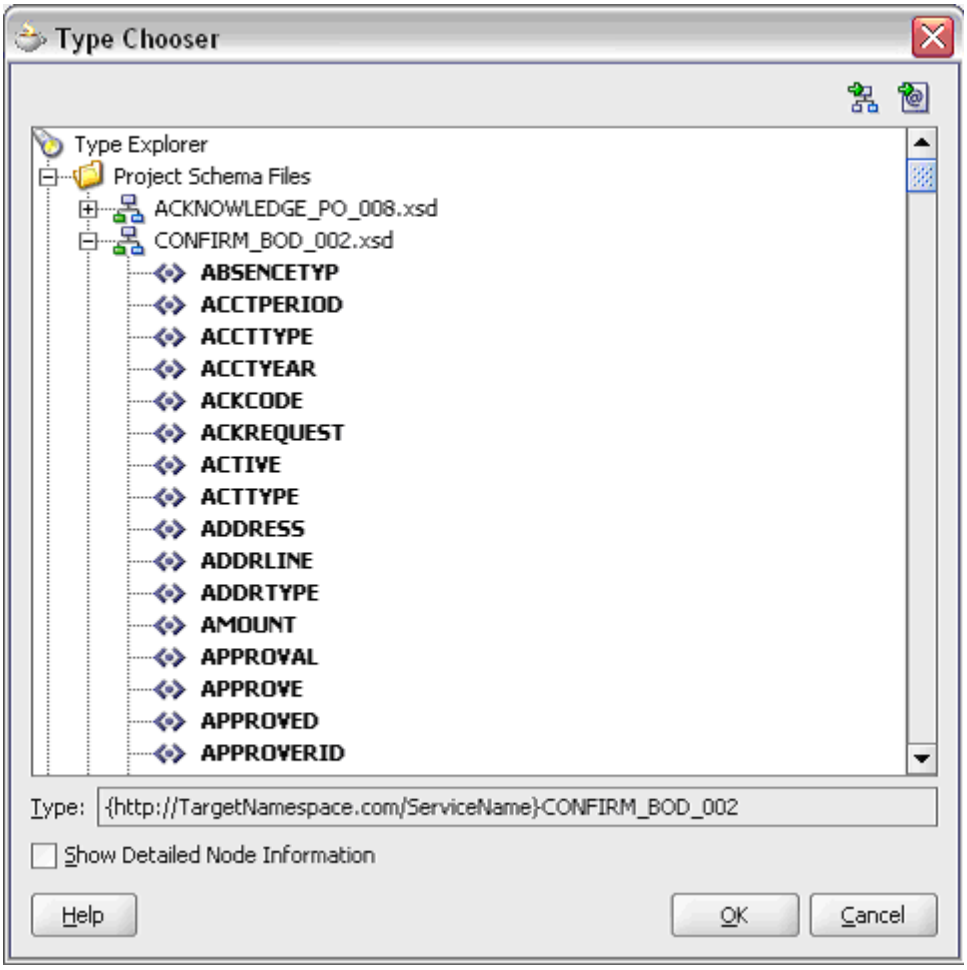
6. For the Directory specified as field, select **Logical Path**. Enter directory path in the Directory for Outgoing Files field, and specify a naming convention for the output file such as `POAck%yyMMddJJmmss%.xml`.

7. Confirm the default write condition: Number of Messages Equals 1. Click **Next**. The Messages dialog box appears.

8. Select **Browse** check box to locate the schema location and schema element.

   The Type Chooser dialog box appears. Expand the **Project Schema Files > CONFIRM_BOD_002.xsd** and select **CONFIRM_BOD_002**.

Type Chooser Dialog



Click **OK** to populate the selected schema location and element.

9. Click **Next** and then **Finish**. The wizard generates the WSDL file corresponding to the partner link. The main Create Partner Link dialog box appears, specifying the new WSDL file `WriteAck.wsdl`.

   Click **Apply** and then **OK** to complete the configuration and create the partner link with the required WSDL settings for the File Adapter Service.

   The `WriteAck` Partner Link appears in the BPEL process diagram.

Adding an Invoke Activity

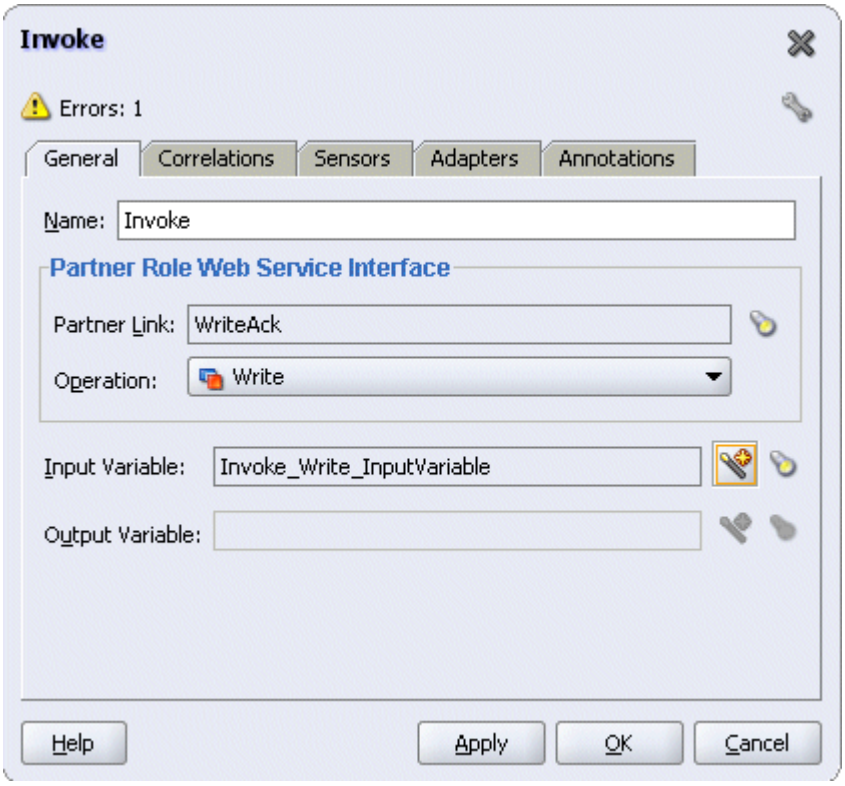
This step is to configure an Invoke activity to send the purchase order acknowledgement that is received from the Receive activity to the `WriteAck` partner link in an XML file.

To add an Invoke activity:

1. In JDeveloper BPEL Designer, drag and drop the **Invoke** activity from the Component Palette into the Activity box of the process diagram, after the **Receive** activity.

2. Link the Invoke activity to the `WriteAck` service. The Invoke activity will send event data to the partner link. The Edit Invoke dialog box appears.

Edit Invoke Dialog



3. Enter a name for the Invoke activity, and then click the **Create** icon next to the **Input Variable** field to create a new variable. The Create Variable dialog box appears.

4. Select **Global Variable**, and then enter a name for the variable. You can also accept the default name. Click **OK**.

   Click **Apply** and then **OK** in the Edit Invoke dialog box to finish configuring the Invoke activity.

   The Invoke activity appears in the process diagram.

Adding an Assign Activity

Use this step to pass the purchase order acknowledgement details from the Receive

activity to the Invoke activity.

To add an Assign activity:

1. In JDeveloper BPEL Designer, drag and drop the **Assign** activity from the Component Palette into the Activity box of the process diagram, between the **Receive** activity and the **Invoke** activity.

2. Double-click the **Assign** activity to access the Edit Assign dialog box.

3. On the Copy Operation tab, click **Create** and then select **Copy Operation** from the menu. The Create Copy Operation window appears.

4. In the From navigation tree, select type Variable. Navigate to **Variable > Process > Variables > Receive_DEQUEUE_InputVariable** and select **CONFIRM_BOD_002**. The XPath field should contain your selected entry.

5. In the To navigation tree, select type Variable. Navigate to **Variable > Process > Variables > Invoke_Write_InputVariable** and select **CONFIRM_BOD_002**. The XPath field should contain your selected entry.

6. Click **OK**.

   Click **Apply** and then **OK** in the Edit Assign dialog box to complete the configuration of the Assign activity.

Deploying and Testing the BPEL Process at Run Time

After creating the BPEL process, you can deploy it to a BPEL server. To ensure that this process is modified or orchestrated appropriately, you can also test the BPEL process by initiating the business process contained in the BPEL process to test the interface integration.

Prerequisites

Before deploying the BPEL process using Oracle JDeveloper, you must ensure that you have established the connectivity between the deign-time environment and the run-time servers including the application server and the integration server.

How to configure the necessary server connection, see Configuring Server Connection, page B-1.

To validate the BPEL process, preform the following run-time tasks:

1. Deploy the BPEL process, page 5-42

   Once you deploy the process to a BPEL server, it becomes available so that you can run the process manually to test it for validation.

2. Manually test the BPEL process, page 5-42

After deploying a BPEL process, you can manage the process from the BPEL console to manually initiate the business process and test the interface integration contained in your BPEL process.

Deploying the BPEL Process

Before manually test the BPEL process, you first need to deploy it to the BPEL server.

To deploy the BPEL process:

1. In the Applications Navigator of JDeveloper BPEL Designer, select the **XMLGOutbound** project.

2. Right-click the project and click **Make** action from the menu.

   Look for any compilation error messages in Message Log.

   Right-click the project and select **Deploy >*Integration Server Connection name*> Deploy to Default Domain** action from the menu.

   For example, you can select **Deploy > BPELServerConn > Deploy to Default Domain** to deploy the process if you have the BPEL Process Manager setup appropriately.

3. Look for 'Build successful' message in Apache Ant – Log to ensure that the BPEL project is compiled and successfully deployed.

Testing the BPEL Process

To validate whether the BPEL process that you created works or not, you need to manually initiate the process after it has been successfully deployed to the BPEL console. You can log on to Oracle E-Business Suite to manually create and book the order as well as generate the order acknowledgement by submitting a Workflow Background Process concurrent request.

Log in to the BPEL console to validate the BPEL process which writes purchase order acknowledgement in an output directory after receiving from the XML Gateway ECX_OUTBOUND queue.

To manually test the BPEL process:

1. Log in to Oracle E-Business Suite with the XML Gateway responsibility.

   This is to ensure that the XML Gateway trading partner is set up correctly so that a purchase order can have a valid customer that has been defined.

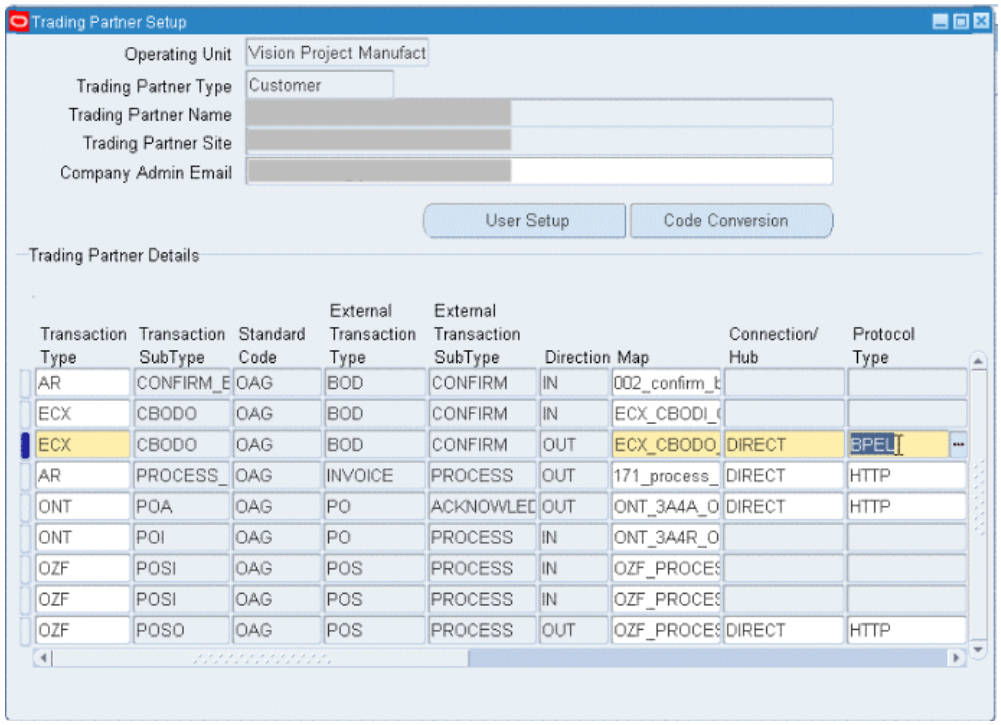
2. Select Define Trading Partner from the navigation menu to access the Trading Partner Setup form.

3. Enter the header values on the Trading Partner Setup form as follows:

- Trading Partner Type: Customer
- Trading Partner Name: For example, Example Inc.
- Trading Partner Site: Enter a trading partner site information. For example, 401 Island Parkway Redwood Shores, CA 94065
- Company Admin Email: Enter a valid email address.

4. Enter the following trading partner details:
   - Transaction Type: ECX
   - Transaction SubType: CBODO
   - Standard Code: OAG
   - External Transaction Type: BOD
   - External Transaction SubType: CONFIRM
   - Direction: Out
   - Map: ECX_CBODO_OAG62_OUT
   - Connection / Hub: DIRECT
   - Protocol Type: BPEL
   - Username: 'operation'
   - Password: enter password twice
   - Protocol Address: 'http:us.example.com'
   - Source Trading Partner Location Code: Example-01

Trading Partner Setup Form



5. Save the trading partner details. Switch responsibility back to Order Management Super User, Vision Operations (USA) and select Customer > Standard from the navigation menu to open the Enter Customer form.

6. Search on the 'Example Inc' in the Name field and click **Find**.

7. Select the Business World with the following information from the search results.
   - Party Number: 2813
   - Customer Number: 1608
   - Account Name: Example Inc.
   - Identifying check box: checked
   - Address: 401 Island Parkway Redwood Shores, CA 94065

8. Select and open this customer information. Enter 'Example-01' in the EDI Location field.

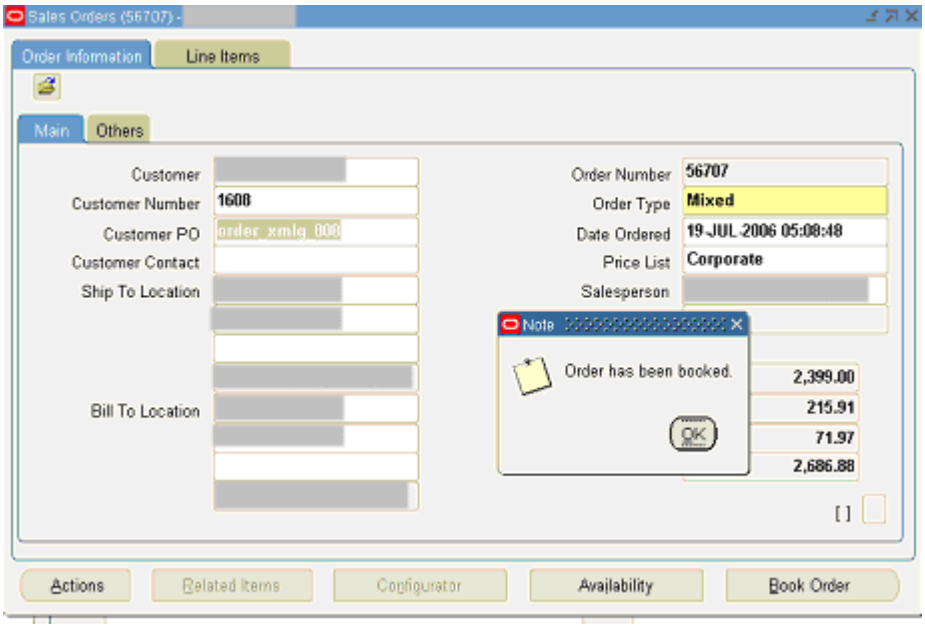
9. In Business Purposes tab, create a new row with the following values:

- Usage: Sold To
- Check on 'Primary' Check box

Save your work.

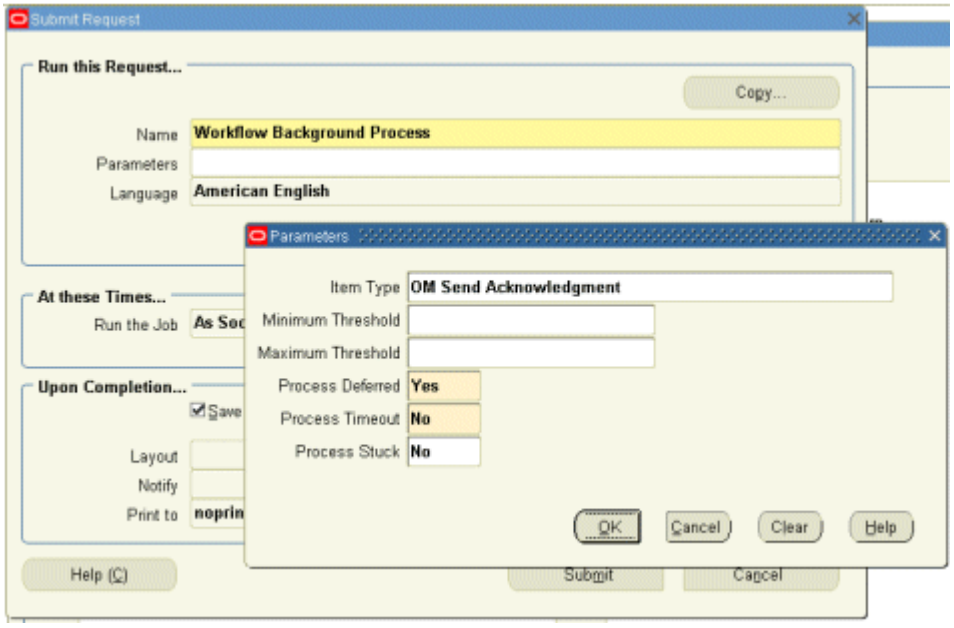
10. Use the following steps to generate acknowledgement for already created order.
    1. Select **Order Returns > Sales Order** to open the Sales Order form.
    2. Retrieve the order that you have created earlier by entering the order ID in the Customer PO field.
    3. Click **Book Order** to book the order.

Booking an Order



11. Switch to the System Administrator responsibility and select **Request > Run**.
12. Select **Single Request** and click **OK**.
13. Enter the following information in the Submit Request form:

Submit Request Form and Parameters Dialog



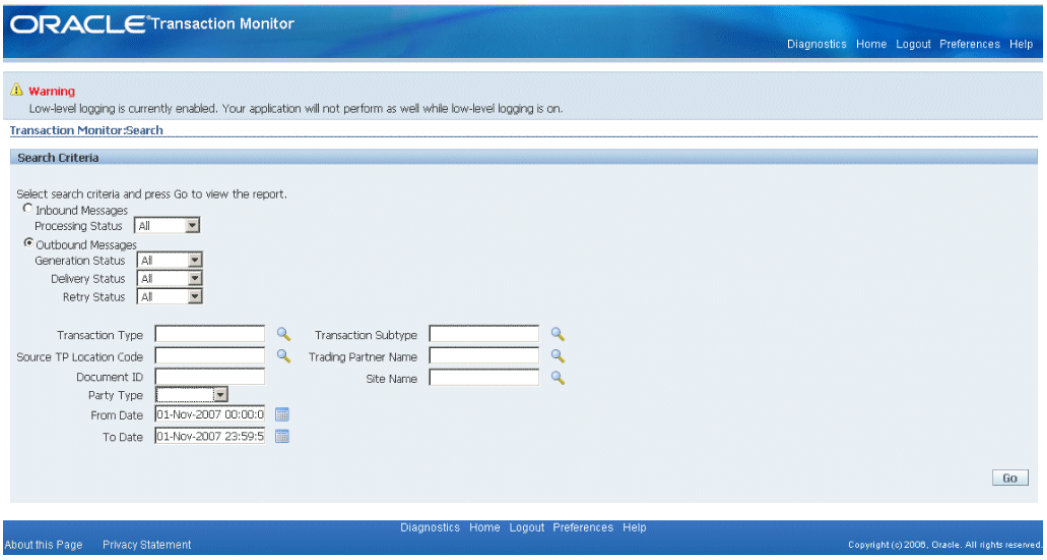
- Name: Workflow Background Process
- Enter the following parameters:
  - Item Type: OM Send Acknowledgement
  - Process Deferred: Y
  - Process Timeout: N
  - Process Stuck: N
- Click **OK**.

14. Click **Submit** to submit the 'send acknowledgement' request.
15. View your request by entering the request ID to ensure its status is 'Success'.

Validating Using Oracle Transaction Monitor

To validate it using Oracle Transaction Monitor, log in to Oracle E-Business Suite with the Workflow Administrator Web Applications responsibility. Select **Transaction Monitor** to open the search page to search for the order.

Transaction Monitor: Search Page



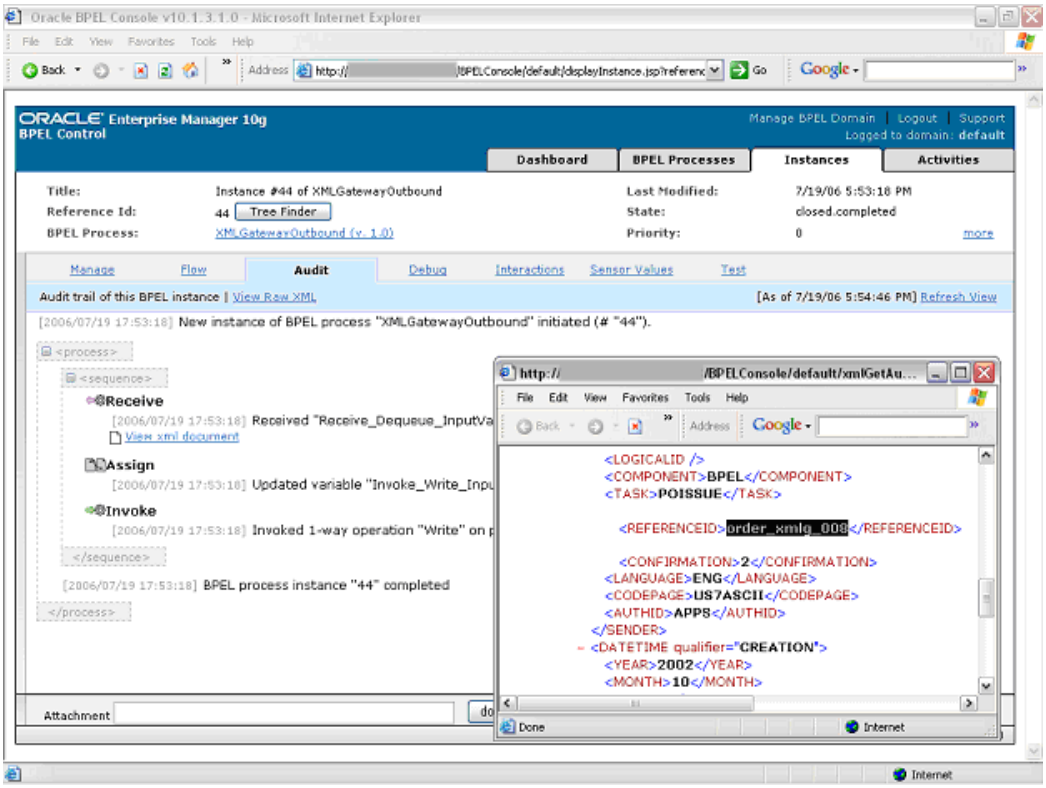
Validating Using Oracle BPEL Console

Log in to Oracle BPEL Console to confirm that the `XMLGOutbound` process has been deployed. This process is continuously polling the ECX_OUTBOUND queue for purchase order acknowledgement.

To verify, select the instance of your deployed process which opens up in the Instances tab of your selected BPEL process.

Click on the Audit Tab to view the Receive activity. Click the **view xml document** link to open the received XML file. Note the Reference ID such as Customer PO.

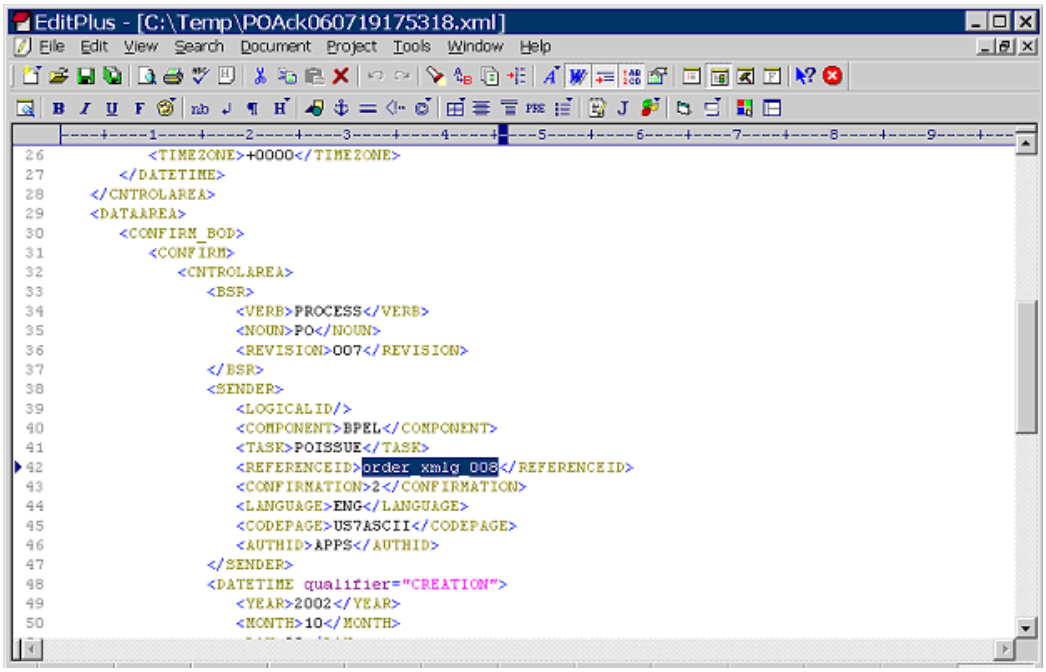
Viewing XML File for the Receive Activity



Go to the directory you specified for the write operation, for example `outputDir` - logical location (typically `c:\temp`) where the File Adapter has placed the file after writing the PO Acknowledgement in an XML file (such as 'POAck060719175318.xml').

Open this 'POAck060719175318.xml' file. You should find the Reference ID as order_xmlg_008 (the order booked) for which the acknowledgement is generated.

PO Acknowledgement XML File



6

Using Business Events Through Subscription Model

Overview

The Oracle Workflow Business Event System (BES) is an application service that leverages the Oracle Advanced Queuing (AQ) infrastructure to communicate business events between systems. The Business Event System consists of the Event Manager and workflow process event activities.

The Event Manager lets you register subscriptions to significant events; event activities representing business events within workflow processes let you model complex business flows or logics within workflow processes.

Events can be raised locally or received from an external system or the local system through AQ. When a local event occurs, the subscribing code is executed in the same transaction as the code that raised the event, unless the subscriptions are deferred.

Oracle E-Business Suite Integrated SOA Gateway supports business events through event subscription. An integration repository administrator can subscribe to a business event from the business event interface details page. The subscription to that event can be enqueued as an out agent. An integration developer can create a BPEL process in Oracle JDeveloper to include the subscribed event at design time and update application data if needed at run time.

To better understand how the subscription model works for business events, detailed tasks at design time and run time are included in this chapter. For the example described in the following sections, we use Oracle JDeveloper 10.1.3.3.0 as a design-time tool to create the BPEL process and use Oracle SOA Suite BPEL server 10.1.3.3.0 for the process deployment.

Using a Business Event in Creating a BPEL Process at Design Time

BPEL Process Scenario

Take a PO XML Raise business event as an example to explain the BPEL process creation.

When a purchase order is created and approved, a Purchase Order Approved business event `oracle.apps.po.evnt.xmlpo` is raised. Since the subscription to this event is created through the interface details page (internally, an event subscription is automatically created for the selected event with `WF_BPEL_QAGENT` as Out Agent), and enqueued in `WF_EVENT_T` structure to Advanced Queue `WF_BPEL_Q`, we will create a BPEL process to first dequeue the subscription from the `WF_BPEL_Q` queue to get the event details. The event details will be passed through BPEL process activities and then will be written in XML file as an output file.

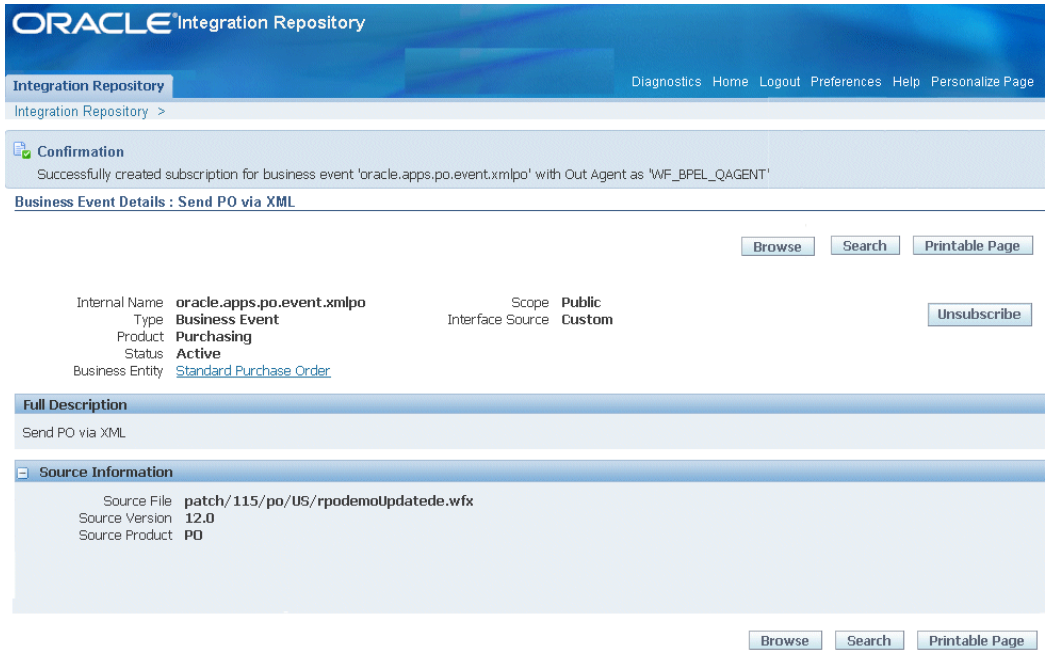
If the BPEL process is successfully executed after deployment, you should get the same purchase order information from the output file once a purchase order is approved.

Prerequisites to Create a BPEL Process Using a Business Event

Integration repository administrators must first subscribe to a business event from the Oracle Integration Repository user interface. Internally, an event subscription is automatically created for that event with `WF_BPEL_QAGENT` as Out Agent.

For example, a business event `oracle.apps.po.event.xmlpo` needs to be subscribed. A confirmation message appears if the event subscription is successfully created.

Business Event Details Page



To subscribe to a business event, the administrators will first locate an event from the Oracle Integration Repository, and then click **Subscribe** in the interface detail page to create the subscription.

For information on how to subscribe to business events, see Subscribing to Business Events, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

> **Note:** If a BPEL process is created with the business event that you have subscribed to it, in order for the subscribed business event to be successfully enqueued to `WF_BPEL_Q` queue, you need to make sure:
>
> - The consumer name must be unique.
> - The BPEL process is deployed before raising the business event.

Once the subscription is created and enqueued, an integration developer can then orchestrate the subscribed event into a meaningful business process in Oracle JDeveloper using BPEL language at design time.

BPEL Process Creation Flow

Based on the PO XML Raise business event scenario, the following design-time tasks are discussed in this chapter:

1. Create a new BPEL project, page 6-5

   Use this step to create a new BPEL project called `GetPurchaseOrder.bpel`.

2. Create a Partner Link for AQ Adapter, page 6-6

   Use this step to dequeue the event details from the `WF_BPEL_Q` queue.

3. Add a Receive activity, page 6-11

   Use the Receive activity to take event details as an input to the Assign activity.

4. Create a Partner Link for File Adapter, page 6-13

   This is to write event details in an XML file as an output file.

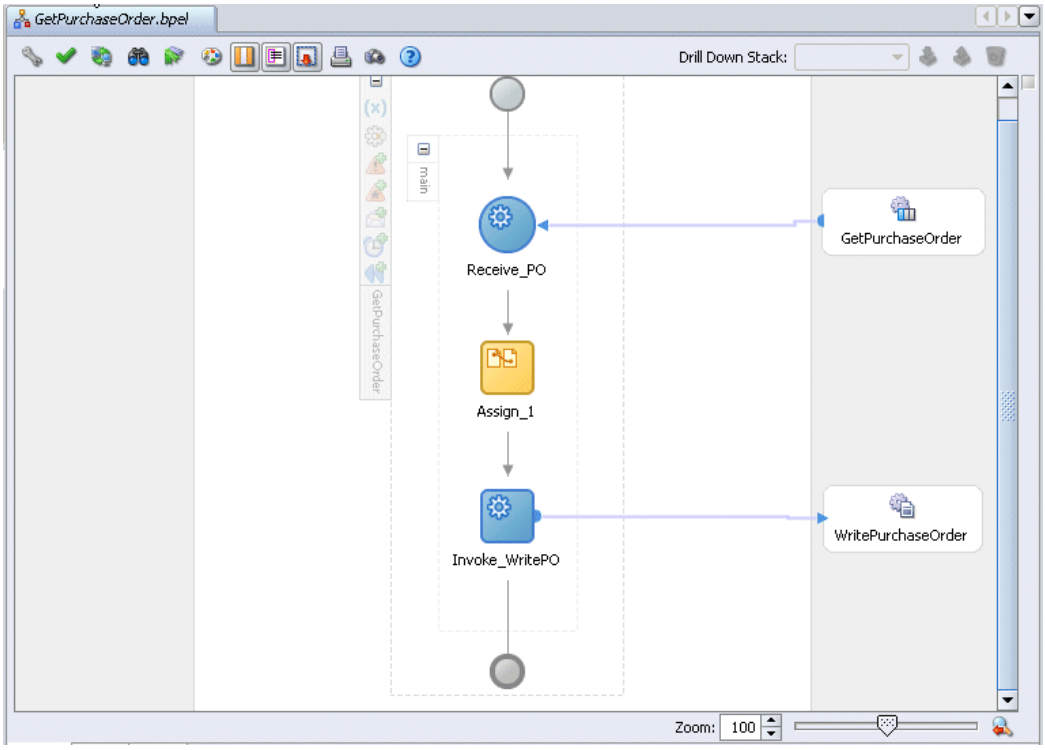
5. Add an Invoke activity, page 6-15

   This is to write business event information to an XML file through invoking the partner link for File Adapter.

6. Add an Assign activity, page 6-17

   Use the Assign activity to take the output from the Receive activity and to provide input to the Invoke activity.

Example of a BPEL Project Using Business Events



For general information and basic concept of a BPEL process, see Understanding BPEL Business Processes, page D-1 and *Oracle BPEL Process Manager Developer's Guide* for details.

Creating a New BPEL Project

Use this step to create a new BPEL project that will contain various BPEL process activities.

To create a new BPEL project:

1. Open JDeveloper BPEL Designer.
2. From the **File** menu, select **New**. The New Gallery dialog box appears.
3. Select **All Technologies** from the **Filter By** box. This produces a list of available categories.
4. Expand the **General** node and then select **Projects**.
5. Select **BPEL Process Project** from the **Items** group.
6. Click **OK**. The BPEL Process Project dialog box appears.

7. In the **Name** field, enter a descriptive name such as `GetPurchaseOrder`.

8. From the Template list, select **Empty BPEL Process** and then select **Use Default Project Settings**.

9. Click **Finish**.

   A new BPEL project is created with the required source files including bpel.xml, using the name you specified (for example, `GetPurchaseOrder.bpel`).

Creating a Partner Link for AQ Adapter

Use this step to create a Partner Link called `GetPurchaseOrder` for AQ Adapter to dequeue the subscription to `oracle.apps.po.evnt.xmlpo` event.

To create a partner link for AQ Adapter to dequeue the event subscription:

1. In JDeveloper BPEL Designer, drag and drop the **AQ Adapter** service from the Component Palette into the Partner Link border area of the process diagram. The Adapter Configuration Wizard appears.

2. Enter a service name in the Service Name dialog box, for example `GetPurchaseOrder`. You can also add an optional description of the service.

3. Click **Next**. The Service Connection dialog box appears.

4. You can use an existing database connection by selecting a database connection from the **Connection** list or define a new database connection by clicking **New** to open the Create Database Connection Wizard.

   Note: You need to connect to the database where Oracle E-Business Suite is running.

   To create a new database connection:

   1. Click **New** to open the Create Database Connection Wizard. Click **Next** and enter an unique connection name and then select a connection type for the database connection. Click **Next**.

   2. Enter an appropriate username and password to authenticate the database connection in the Authentication dialog box. Click **Next**

   3. Specify the following information in the Connection dialog box:

      - Driver: Thin

      - Host Name: Enter the host name for the database connection. For example, `myhost01.example.com`.

- JDBC Port: Enter JDBC port number (such as `1521`) for the database connection.
- SID: Specify an unique SID value (such as `sid01`)for the database connection.

4. Click **Next** to test your database connection.

   The status message `"Success!"` indicates a valid connection.

5. Click **Next** to return to the Service Connection dialog box providing a summary of the database connection.

5. The JNDI (Java Naming and Directory Interface) name corresponding to the database connection you specified appears automatically in the **JNDI Name** field of the Service Connection dialog box. Alternatively, you can enter a different JNDI name.

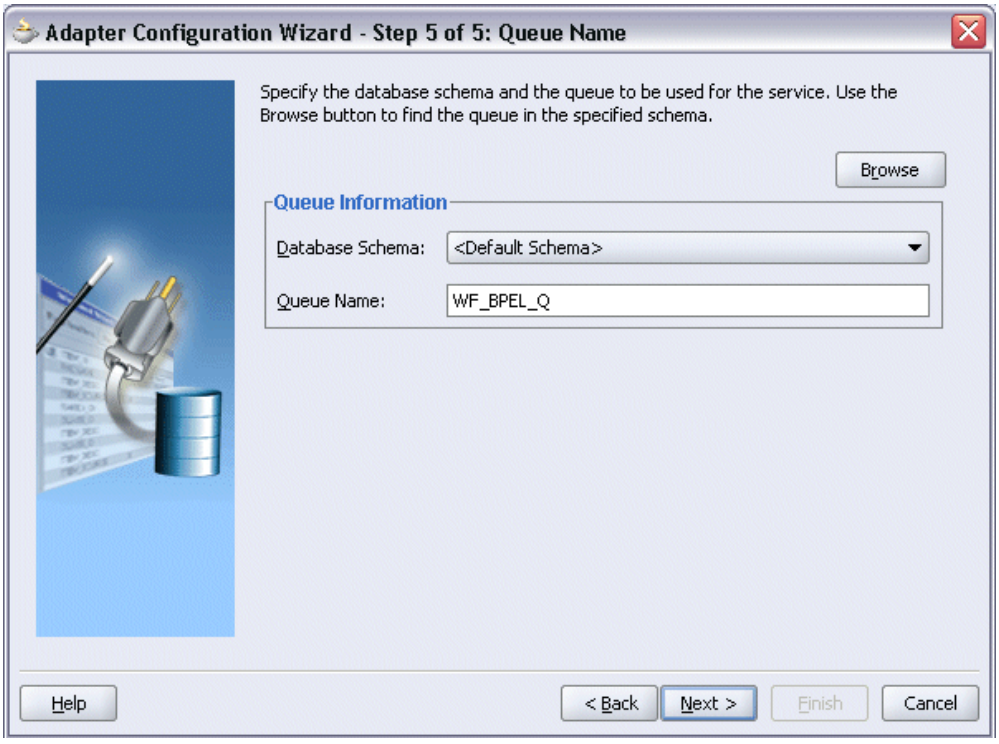
6. Click **Next** to open Operation dialog box.

   Select **Dequeue** radio button and this selected value is also populated in the Operation Name field.

7. Click **Next** to open the Queue Name dialog box.

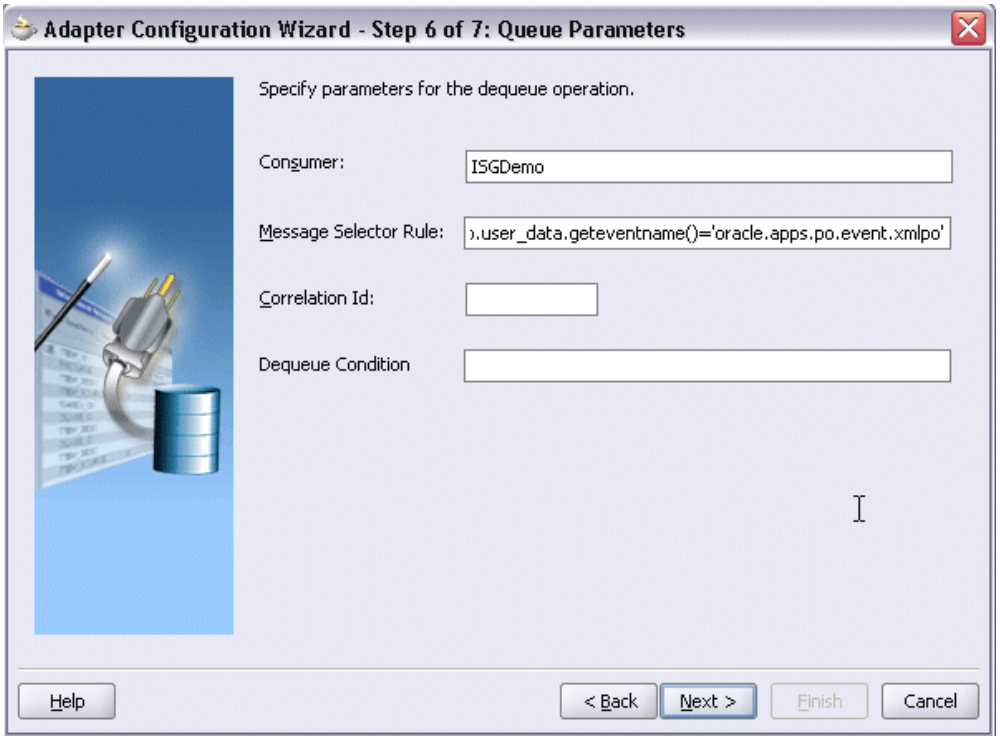
   Select Default Schema as the Database Schema field. Enter 'WF_BPEL_Q' as the Queue Name field.

Queue Name Dialog



8. Click **Next** to open the Queue Parameter dialog box.

Queue Parameters Dialog



Enter the following information:

- Enter an unique consumer name.

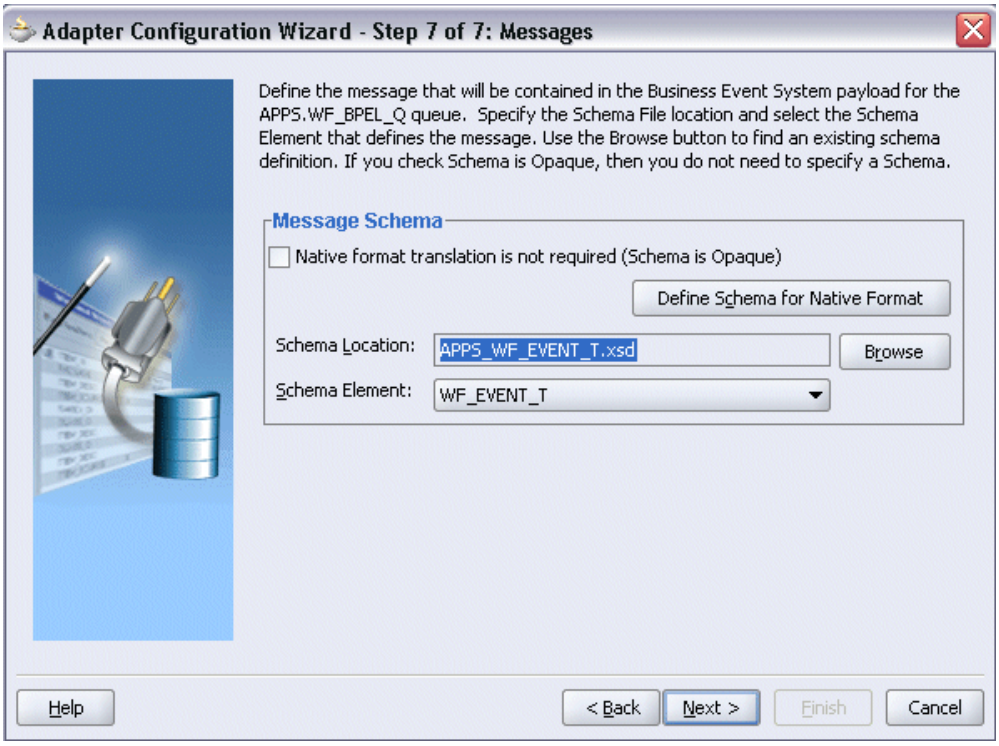
  Important: In order for the subscribed business event to be successfully enqueued to `WF_BPEL_Q` queue, the consumer name must be unique.

- Enter message selector rule information (such as `tab.user_data.geteventname()='oracle.apps.po.evnt.xmlpo'`).

9. Click **Next**. The Messages dialog box opens where you can define the message that will be contained in the Business Event System payload for the APPS.WF_BPEL_Q queue.

   Click **Browse** to open the Type Chooser window to select APPS_WF_EVENT_T.xsd as the Schema Location and WF_EVENT_T as the Schema Element.

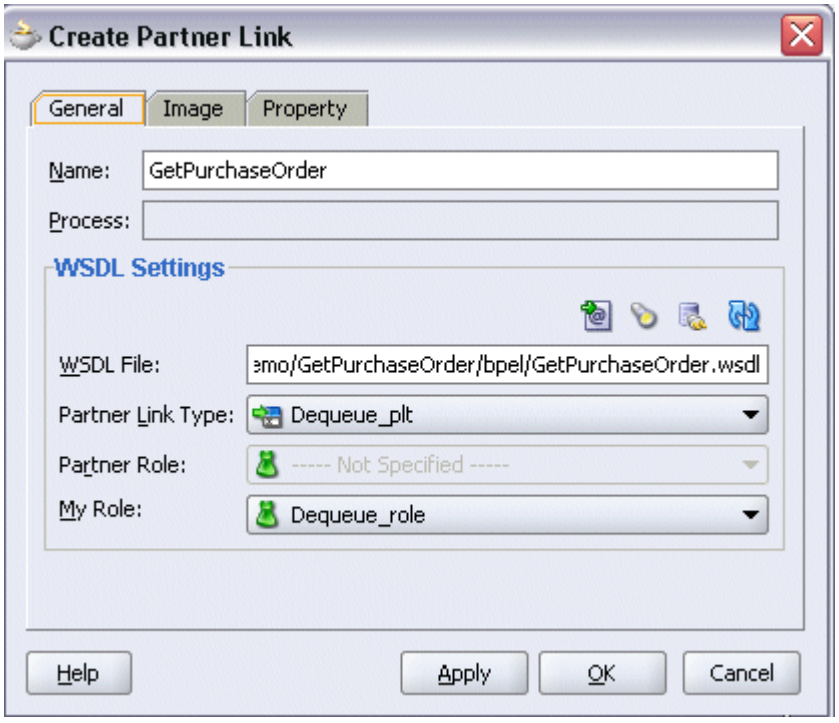
Messages Dialog with Selected Message Schema



10. Click **Next** to proceed to the Finish dialog box to confirm that you have finished defining the AQ Adapter for the `GetPurchaseOrder` service.

11. Click **Finish**. The wizard generates the WSDL file corresponding to the `GetPurchaseOrder` service.

Create Partner Link Dialog



Click **Apply** and then **OK** to complete the partner link configuration. The partner link is created with the required WSDL settings, and is represented in the BPEL project by a new icon in the border area of the process diagram.

Adding a Receive Activity

This step is to configure a Receive activity to receive XML data from the partner link `GetPurchaseOrder` that you configured for the AQ adapter service for the business event.

The XML data received from the Receive activity is used as an input variable to the Assign activity that will be created in the next step.

To add a Receive activity to obtain Purchase Order XML data:

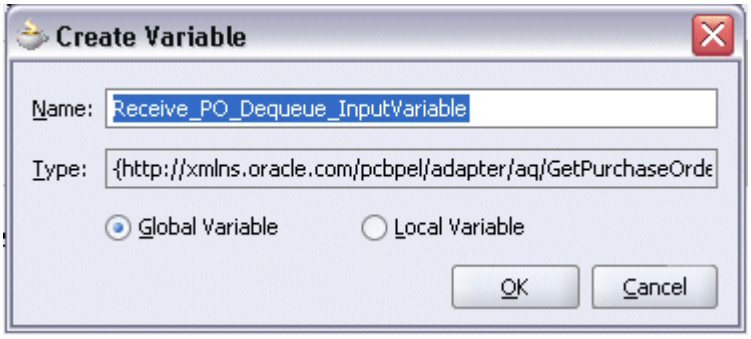
1. In JDeveloper BPEL Designer, drag and drop the **Receive** activity from the **BPEL Activities** section of the Component Palette into the Activity box of the process diagram.

2. Link the Receive activity to the `GetPurchaseOrder` partner link. The Receive activity will take event data from the partner link. The Edit Receive dialog box appears.

3. Enter a name for the receive activity such as 'Receive_PO' and then click the **Create**

icon next to the **Variable** field to create a new variable. The Create Variable dialog box appears.

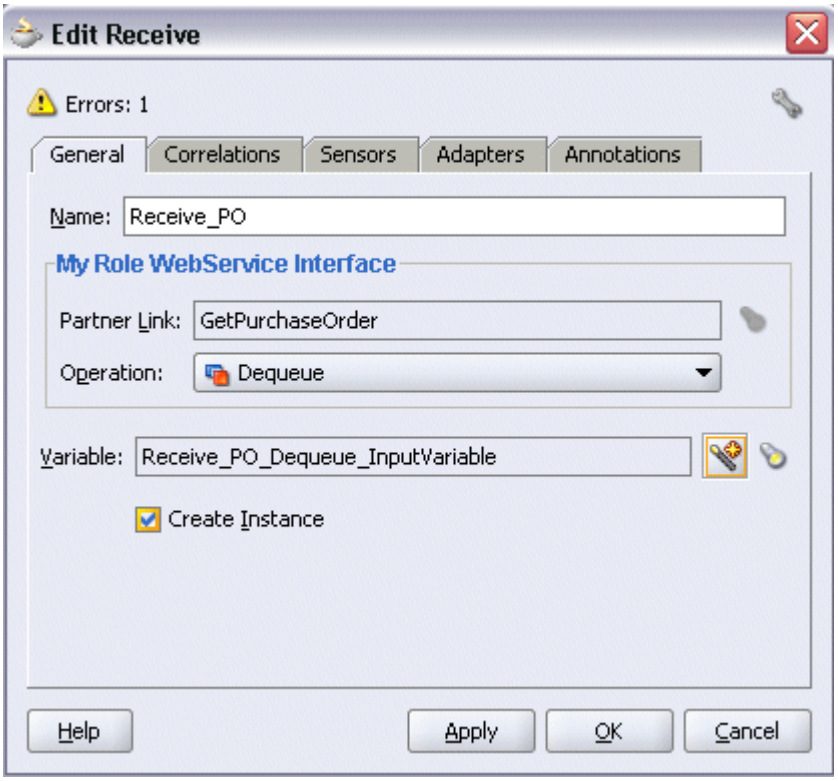
Create Variable Dialog



4. Select **Global Variable** and then enter a name for the variable. You can accept the default name. Click **OK** to return to the Edit Receive dialog box.

5. Select **Create Instance** check box then click **Apply** and then **OK** to finish configuring the Receive activity.

Edit Receive Dialog



The Receive activity appears in the BPEL process diagram.

Adding a Partner Link for File Adapter

Use this step to configure a business event by writing the event data to an XML file.

To add a Partner Link for File Adapter:

1. In JDeveloper BPEL Designer, drag and drop the **File Adapter** service from the **Adapter Service** section of the Component Palette into the Partner Link area of the process diagram. The Adapter Configuration wizard appears.

2. The Service Name dialog box appears.

3. Enter a name for the file adapter service such as `WritePurchaseOrder`. You can add an optional description of the service.

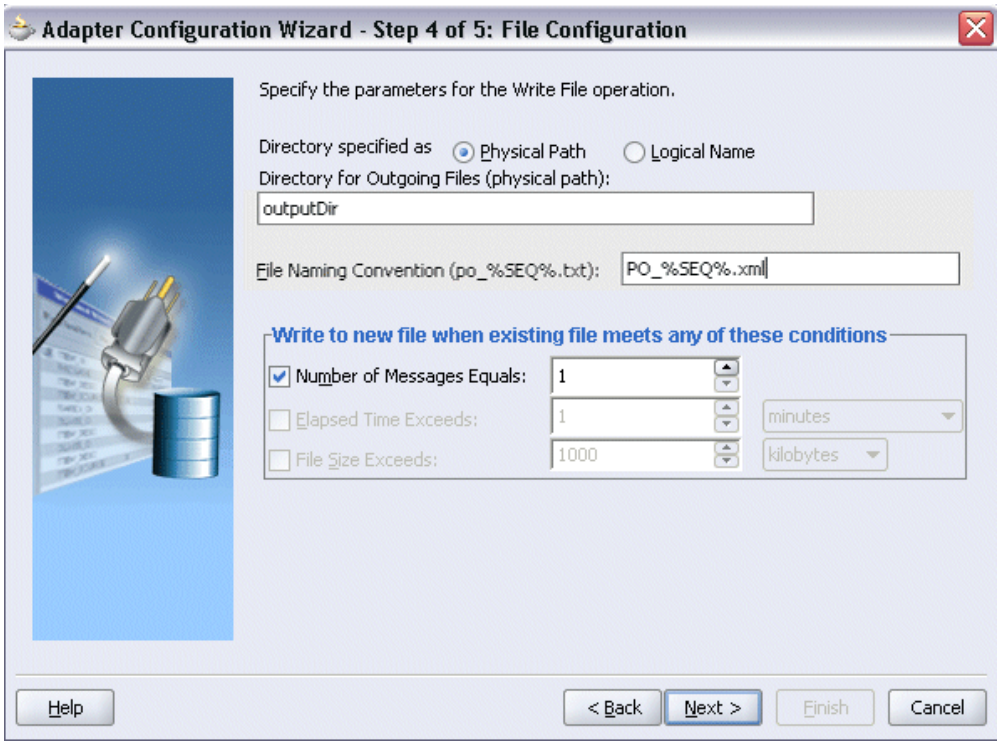
4. Click **Next** and the Operation dialog box appears.

Operation Dialog

5. Specify the operation type, for example **Write File**. This automatically populates the **Operation Name** field.

   Click **Next** to access the File Configuration dialog box.

File Configuration Dialog



6. For the Directory specified as field, select **Physical Path**. Enter directory path in the Directory for Outgoing Files field, and specify a naming convention for the output file such as `PO_%SEQ%.xml`.

7. Confirm the default write condition: Number of Messages Equals 1. Click **Next**. The Messages dialog box appears.

8. Select **Native format translation is not required (Schema is Opaque)** check box.

9. Click **Next** and then **Finish**. The wizard generates the WSDL file corresponding to the partner link. The main Create Partner Link dialog box appears, specifying the new WSDL file `WritePurchaseOrder.wsdl`.

   Click **Apply** and then **OK** to complete the configuration and create the partner link with the required WSDL settings for the File Adapter Service.

   The `WritePurchaseOrder` Partner Link appears in the BPEL process diagram.

Adding an Invoke Activity

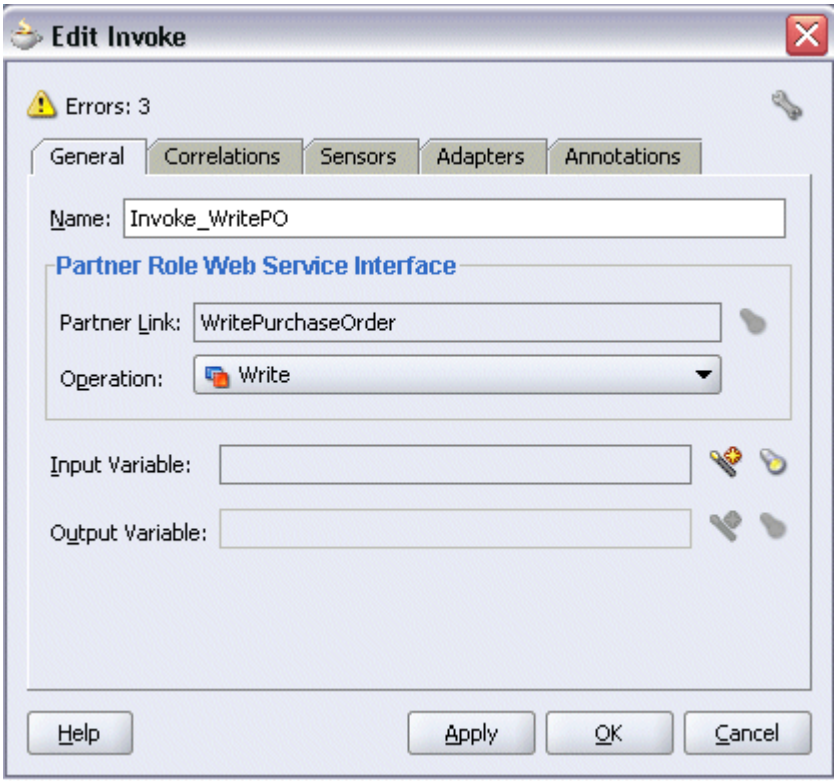
This step is to configure an Invoke activity to write the purchase order approved event details that is received from the Receive activity to the `WritePurchaseOrder` partner link in an XML file.

To add an Invoke activity:

1. In JDeveloper BPEL Designer, drag and drop the **Invoke** activity from the Component Palette into the Activity box of the process diagram, after the **Receive** activity.

2. Link the Invoke activity to the `WritePurchaseOrder` service. The Invoke activity will send event data to the partner link. The Edit Invoke dialog box appears.

Edit Invoke Dialog



3. Enter a name for the Invoke activity, and then click the **Create** icon next to the **Input Variable** field to create a new variable. The Create Variable dialog box appears.

4. Select **Global Variable** and then enter a name for the variable. You can also accept the default name. Click **OK** to close the Create Variable dialog box.

   Click **Apply** and then **OK** in the Edit Invoke dialog box to finish configuring the Invoke activity.

   The Invoke activity appears in the process diagram.

Adding an Assign Activity

Use this step to pass the purchase order approved event details from the Receive activity to the Invoke activity.

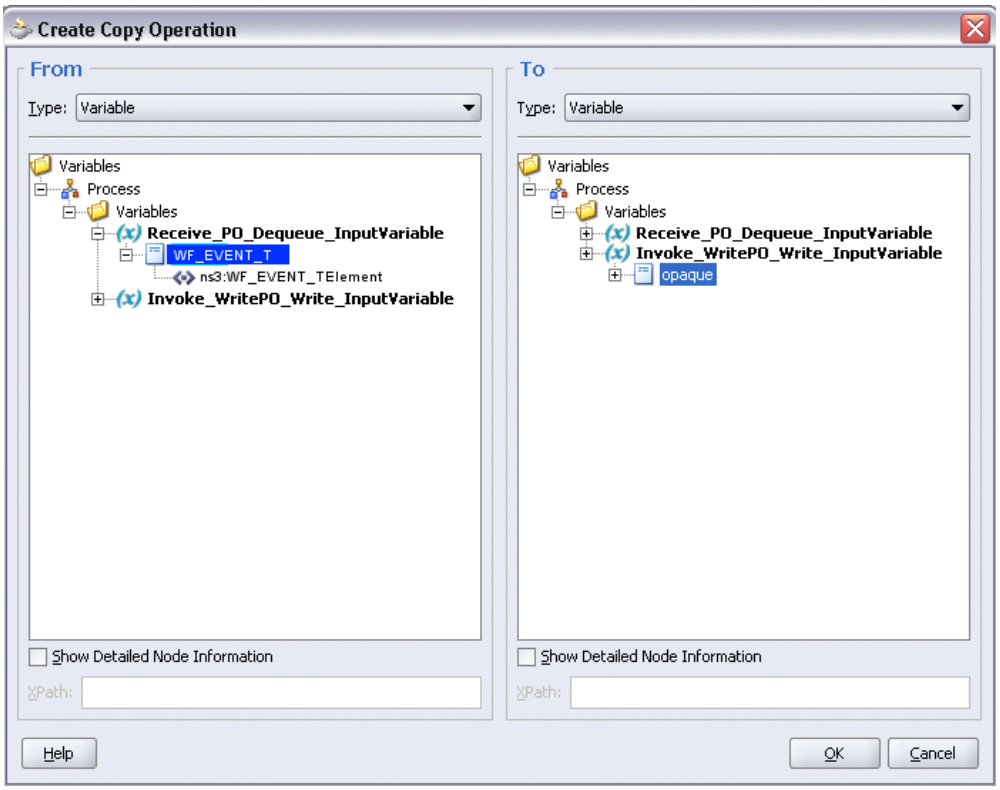
To add an Assign activity:

1. In JDeveloper BPEL Designer, drag and drop the **Assign** activity from the Component Palette into the Activity box of the process diagram, between the **Receive** activity and the **Invoke** activity.

2. Double-click the **Assign** activity to access the Edit Assign dialog box.

3. On the Copy Operation tab, click **Create** and then select **Copy Operation** from the menu. The Create Copy Operation dialog box appears.

Create Copy Operation Dialog



4. In the From navigation tree, select type Variable. Navigate to **Variable > Process > Variables > Receive_PO_DEQUEUE_InputVariable > WF_EVENT_T** and select **ns3:WF_EVENT_T** element. The XPath field should contain your selected entry.

5. In the To navigation tree, select type Variable. Navigate to **Variable > Process >**

Variables > Invoke_WritePO_Write_InputVariable > WF_EVENT_T and select **Opaque**. The XPath field should contain your selected entry.

6. Click **OK** to close the Create Copy Operation dialog box.

   Click **Apply** and then **OK** in the Edit Assign dialog box to complete the configuration of the Assign activity.

Deploying and Testing the BPEL Process at Run Time

After creating a BPEL process with the subscribed event, you can deploy it to a BPEL server if needed. To ensure that this process is modified or orchestrated appropriately, you can also test the BPEL process by initiating the business process contained in the BPEL process to test the interface integration.

Prerequisites

Before deploying the BPEL process using Oracle JDeveloper, you must ensure that you have established the connectivity between the deign-time environment and the run-time servers including the application server and the integration server.

How to configure the necessary server connection, see Configuring Server Connection, page B-1.

To validate the BPEL process, preform the following run-time tasks:

1. Deploy the BPEL process, page 6-18

   Once you deploy the process to a BPEL server, it becomes available so that you can run the process manually to test it for validation.

2. Manually initiate the BPEL process, page 6-19

   After deploying a BPEL process, you can manage the process from the BPEL console to manually initiate the business process and test the interface integration contained in your BPEL process.

Deploying the BPEL Process

Before manually test the BPEL process, you first need to deploy it to the BPEL server.

To deploy the BPEL process:

1. In the Applications Navigator of JDeveloper BPEL Designer, select the **GetPurchaseOrder** project.

2. Right-click the project and click **Make** action from the menu.

   Look for any compilation error messages in Message Log.

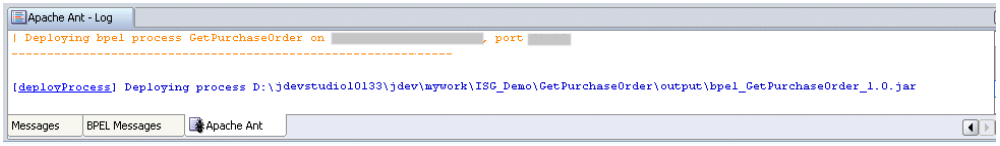
   Right-click the project and select **Deploy >*Integration Server Connection name*>**

Deploy to Default Domain action from the menu.

For example, you can select **Deploy > BPELServerConn > Deploy to Default Domain** to deploy the process if you have the BPEL Process Manager setup appropriately.

3. Look for 'Build successful' message in Apache Ant – Log to ensure that the BPEL project is compiled and successfully deployed.

Compilation and Deployment Message Logs



Testing the BPEL Process

To validate whether the BPEL process that you created works or not, you need to manually initiate the process after it has been successfully deployed to the BPEL console. Therefore, the validation starts with the BPEL console to ensure that you can find the deployed BPEL process listed in the console. Then, you can log on to Oracle E-Business Suite to manually initiate the purchase order approval and acknowledgement processes and to confirm that the relevant event is raised and the updated purchased order details is also written in the XML file.

To manually test the BPEL process:

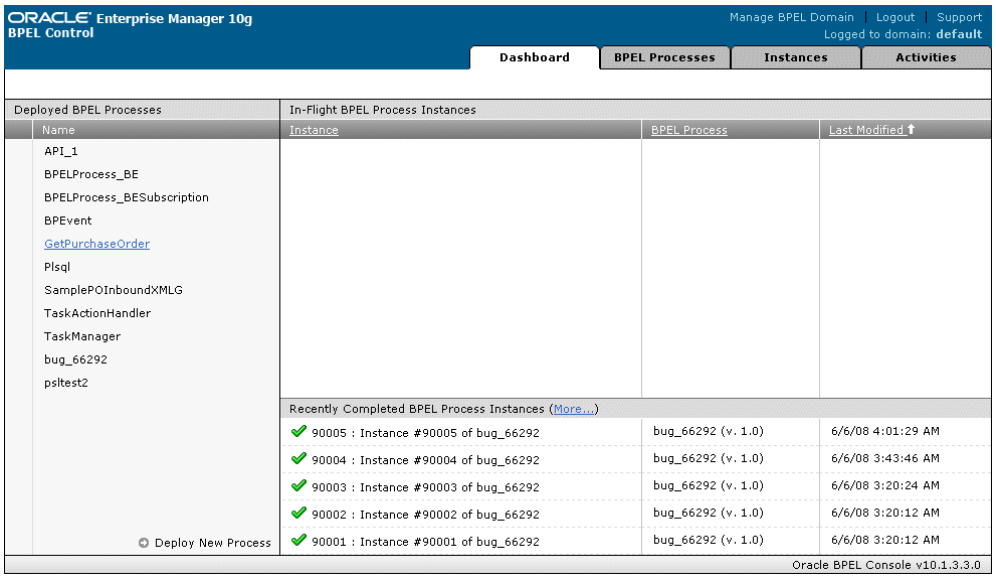
1. Log in to Oracle Application Server 10g BPEL Console (`http://<soaSuiteServerHostName>:<port>/BPELConsole`). The BPEL Console login page appears.

2. Enter the username and password and click **Login**.

   The Oracle Enterprise Manager 10g BPEL Control appears.

3. In the BPEL Control console, confirm that `GetPurchaseOrder` has been deployed.

Oracle Enterprise Manager BPEL Control Console



4. Log in to Oracle E-Business Suite with the XML Gateway responsibility.

   This is to ensure that the XML Gateway trading partner is set up correctly so that a purchase order can have a valid supplier that has been defined.

5. Select Define Trading Partner from the navigation menu to access the Trading Partner Setup window.

6. Enter the header values on the Trading Partner Setup form as follows:

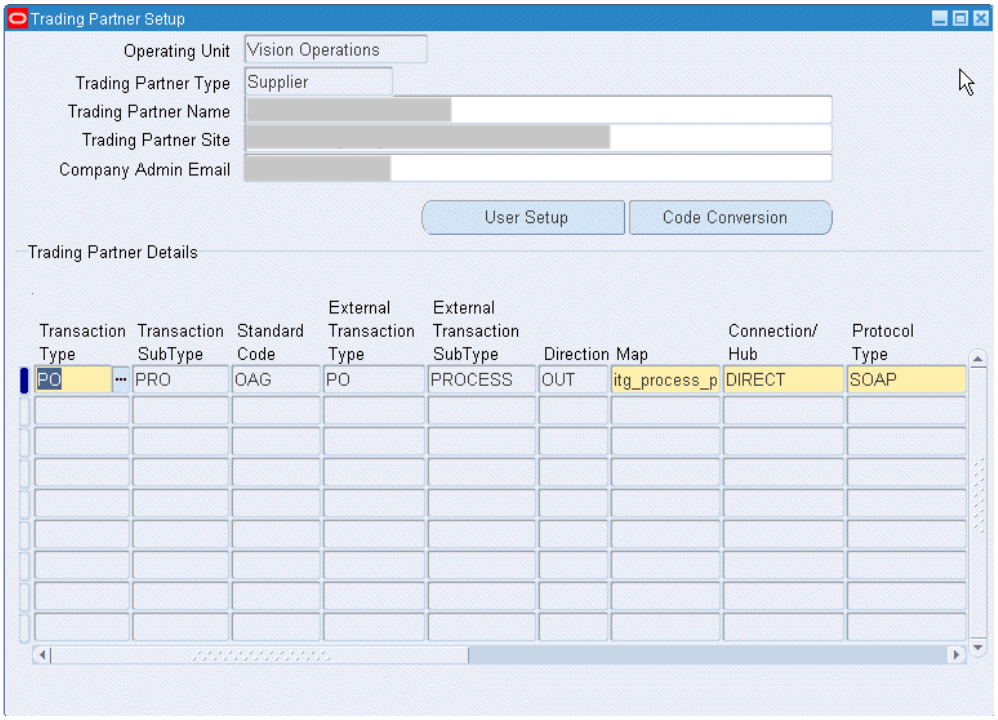
   - Trading Partner Type: Supplier
   - Trading Partner Name: For example, Example Inc
   - Trading Partner Site: Enter a trading partner site information.
   - Company Admin Email: Enter a valid email address.

7. Enter the following trading partner details:

   - Transaction Type: PO
   - Transaction SubType: PRO
   - Standard Code: OAG
   - External Transaction Type: PO

- External Transaction SubType: Process
- Direction: Out
- Map: itg_process_po_007_out
- Connection / Hub: DIRECT
- Protocol Type: SOAP

Trading Partner Setup Form



8. Save the trading partner details. Switch responsibility back to Purchasing, Vision Operations (USA) and select Purchase Order from the navigation menu.

9. Create a purchase order with the header values reflecting the trading partner you previously defined in the Purchase Order window:
   - Supplier: Enter a supplier information.
   - Site: Select a site information.

10. On the Lines tab, enter a data row with the following values:
    - Type: Goods

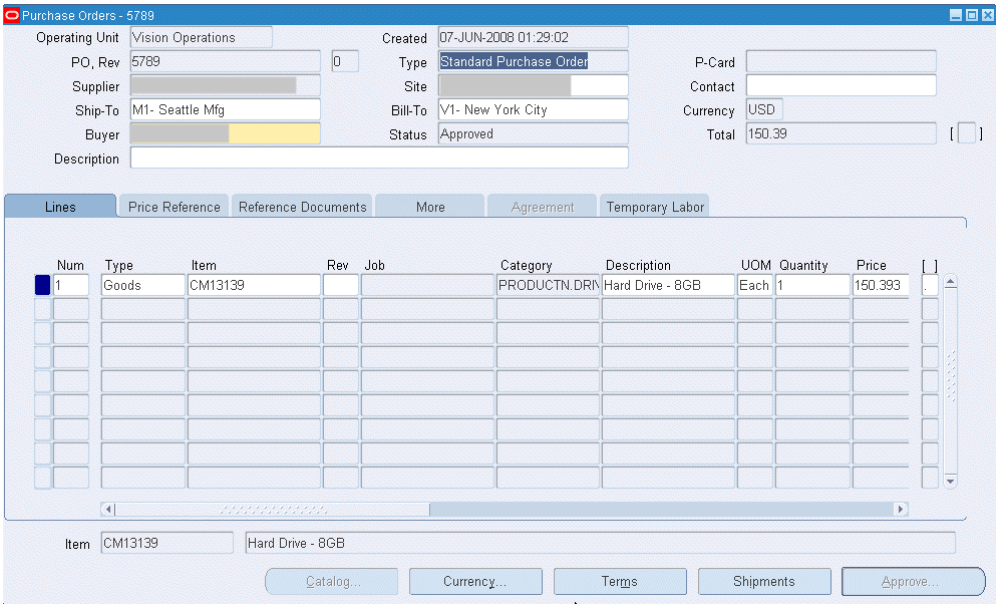
- Item: CM13139
- Quantity: 1
- Description: Hard Drive - 8GB
- Promised: Enter any future date in the format of dd-mmm-yyyy (such as 23-JUN-2008)

11. Save your purchase order. The status of the purchase order is 'Incomplete'.

    Note: Because the trading partner is set up and valid, the transmission method is automatically set to **XML**.

12. Click **Approve** to approve the purchase order.

Purchase Orders Form



The status of the purchase order is now changed to 'Approved'. For future reference, note the value of the PO, Rev field (for example, the PO number 5789).

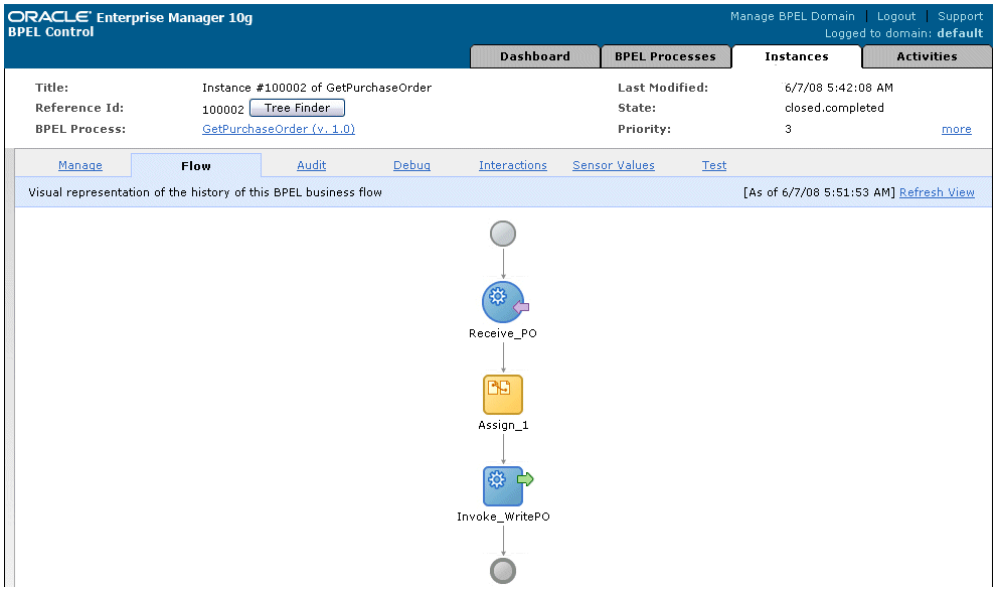
Once the purchase order is approved, the business event `oracle.apps.po.event.xmlpo` is raised.

13. Log in to Oracle BPEL Process Manager, and return to the BPEL Console to confirm that the `GetPurchaseOrder` process has been completed.

    To verify, select the instance of your deployed process which opens up in the

Instances tab of your selected BPEL process.

Instances Tab with the Selected BPEL Process Flow



14. Double-click the Receive activity in the BPEL process diagram and click the View XML document link to open the XML file. Note that the purchase order (number 5789) has been received.

Receive XML File to Verify the Purchase Order Number

```
<?xml version="1.0" encoding="UTF-8" ?>
- <WF_EVENT_T xmlns="http://xmlns.oracle.com/xdb/APPS/GetPurchaseOrder/GetPurchaseOrder">
    <PRIORITY xmlns="">50</PRIORITY>
    <SEND_DATE xmlns="">2008-06-07T05:36:41.000-07:00</SEND_DATE>
    <RECEIVE_DATE xmlns="">2008-06-07T05:38:30.000-07:00</RECEIVE_DATE>
    <CORRELATION_ID xmlns="" />
  - <PARAMETER_LIST xmlns="">
    - <PARAMETER_LIST_ITEM>
        <NAME>APPLICATION_ID</NAME>
        <VALUE>201</VALUE>
      </PARAMETER_LIST_ITEM>
    - <PARAMETER_LIST_ITEM>
        <NAME>DOCUMENT_DIRECTION</NAME>
        <VALUE>OUT</VALUE>
      </PARAMETER_LIST_ITEM>
    - <PARAMETER_LIST_ITEM>
        <NAME>DOCUMENT_NO</NAME>
        <VALUE>5789</VALUE>
      </PARAMETER_LIST_ITEM>
    - <PARAMETER_LIST_ITEM>
        <NAME>ECX_DEBUG_LEVEL</NAME>
        <VALUE>0</VALUE>
      </PARAMETER_LIST_ITEM>
    - <PARAMETER_LIST_ITEM>
        <NAME>ECX_DOCUMENT_ID</NAME>
        <VALUE>5789:0:204</VALUE>
      </PARAMETER_LIST_ITEM>
    - <PARAMETER_LIST_ITEM>
        <NAME>ECX_PARAMETER1</NAME>
        <VALUE />
      </PARAMETER_LIST_ITEM>
    - <PARAMETER_LIST_ITEM>
        <NAME>ECX_PARAMETER2</NAME>
        <VALUE>0</VALUE>
```

15. Examine the Assign and Invoke activities as well for the event raised and document number.

16. Go to the directory you specified for the write operation, for example `outputDir` (typically `c:\temp`). Open the output file (for example `PO_1.xml`), and confirm that the order number is the same as that of the approved purchase order.

Output XML File to Confirm the Order Number

```
<PRIORITY xmlns="">50</PRIORITY>
<SEND_DATE xmlns="">2008-06-07T05:36:41.000-07:00</SEND_DATE>
<RECEIVE_DATE xmlns="">2008-06-07T05:38:30.000-07:00</RECEIVE_DATE>
<CORRELATION_ID xmlns=""/>
<PARAMETER_LIST xmlns="">
   <PARAMETER_LIST_ITEM>
      <NAME>APPLICATION_ID</NAME>
      <VALUE>201</VALUE>
   </PARAMETER_LIST_ITEM>
   <PARAMETER_LIST_ITEM>
      <NAME>DOCUMENT_DIRECTION</NAME>
      <VALUE>OUT</VALUE>
   </PARAMETER_LIST_ITEM>
   <PARAMETER_LIST_ITEM>
      <NAME>DOCUMENT_NO</NAME>
      <VALUE>5789</VALUE>
   </PARAMETER_LIST_ITEM>
   <PARAMETER_LIST_ITEM>
      <NAME>ECX_DEBUG_LEVEL</NAME>
      <VALUE>0</VALUE>
   </PARAMETER_LIST_ITEM>
   <PARAMETER_LIST_ITEM>
      <NAME>ECX_DOCUMENT_ID</NAME>
      <VALUE>5789:0:204</VALUE>
   </PARAMETER_LIST_ITEM>
   <PARAMETER_LIST_ITEM>
      <NAME>ECX_PARAMETER1</NAME>
      <VALUE/>
   </PARAMETER_LIST_ITEM>
   <PARAMETER_LIST_ITEM>
      <NAME>ECX_PARAMETER2</NAME>
      <VALUE>0</VALUE>
   </PARAMETER_LIST_ITEM>
   <PARAMETER_LIST_ITEM>
      <NAME>ECX_PARAMETER3</NAME>
      <VALUE>1318:50578:201</VALUE>
   </PARAMETER_LIST_ITEM>
   <PARAMETER_LIST_ITEM>
      <NAME>ECX_PARAMETER4</NAME>
      <VALUE>97094</VALUE>
   </PARAMETER_LIST_ITEM>
   <PARAMETER_LIST_ITEM>
      <NAME>ECX_PARAMETER5</NAME>
      <VALUE>204</VALUE>
   </PARAMETER_LIST_ITEM>
   <PARAMETER_LIST_ITEM>
      <NAME>ECX_PARTY_ID</NAME>
```

7

Using Concurrent Programs

Overview

A concurrent program is an instance of an execution file with associated parameters. Concurrent programs use a concurrent program executable to locate the correct execution file. The execution file can be an operating system file or database stored procedure which contains your application logic (such as PL/SQL, Java). Several concurrent programs may use the same execution file to perform their specific tasks, each having different parameter defaults.

The concurrent program can be exposed as a Web service based integration interface. An integration repository administrator can further deploy a generated service from Oracle Integration Repository to the application server.

This deployed service can be exposed to customers through service provider and invoked through any of the Web service clients.

For example, an integration developer can take a deployed Web service WSDL URL and directly use it to define a partner link for the Web service that a BPEL process connects to in order to perform tasks, or carry information between the Web service and the BPEL process.

Detailed information on how to create a BPEL process to invoke the Web service and use it to update Oracle E-Business Suite is discussed in this chapter. For the example described in the following sections, we use Oracle JDeveloper 10.1.3.3.0 as a design-time tool to create the BPEL process and use Oracle SOA Suite BPEL server 10.1.3.3.0 for the process deployment.

Using Concurrent Program WSDLs at Design Time

BPEL Process Scenario

This example uses Departure Shipment Notice Outbound `WSHDSNO` concurrent program to explain the BPEL process creation.

When a shipment notice generation request is received as an input to the BPEL process, a sales order information including header and line items are read by a File Adapter. The sales order data is then passed through to create a departure shipment notice (DSNO). The shipment notice creation document number will be passed back to the request application.

If the BPEL process is successfully executed after deployment, you should be able to validate if the generated shipment notice has correct trading partner information as described in the sales order.

Prerequisites to Create a BPEL Process Using a Concurrent Program Web Service

Before performing design-time tasks for concurrent programs, you need to ensure the following tasks are in place:

- An integration repository administrator needs to successfully deploy the generated concurrent program web service to the application server.
- An integration developer needs to locate and record the deployed WSDL URL for the concurrent program exposed as a SOAP service.
- SOAHeader elements should be populated in order to run the concurrent program for SOAP request

Deploying the Concurrent Program WSDL URL

An integration repository administrator must first create a Web service for the selected interface definition, and then deploy the service from Oracle Integration Repository to the application server.

For example, the administrator must perform the following steps before letting integration developers use the deployed WSDL in creating a BPEL process:

1. To generate a web service, locate the interface definition first from the Oracle Integration Repository (such as Departure Shipment Notice Outbound `WSHDSNO` concurrent program) and click **Generate** in the SOAP Web Service tab of the interface details page. For detailed instruction on how to generate a web service, see Generating SOAP Web Services, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.
2. To deploy a generated SOAP service, select at least one authentication type and click **Deploy** in the SOAP Web Service tab of the interface details page to deploy the service.

   Once the service is successfully deployed, the selected authentication type(s) will be displayed along with 'Deployed' Web Service Status. For more information on securing web services with authentication types, see Managing Web Service Security, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

   For detailed instruction on how to deploy a web service, see Deploying and Undeploying SOAP Web Services, *Oracle E-Business Suite Integrated SOA Gateway*

Implementation Guide.

Searching and Recording WSDL URL

Apart from the required tasks performed by the administrator, an integration developer also needs to log on to the system to locate and record the deployed SOAP service WSDL URL for the interface (such as `WSHDSNO` concurrent program) that needs to be orchestrated into a meaningful business process in Oracle JDeveloper using BPEL language.

This WSDL information will be used directly for a partner link during the BPEL process creation at design time.

For information on how to search for an interface and review the interface details, see Searching and Viewing Integration Interfaces, page 2-2.

Setting Variables in SOAHeader for SOAP Request

You must populate certain variables in the BPEL process for SOAHeader elements to pass values that may be required to set application context during service execution. These SOAHeader elements for concurrent program interface type are *`Responsibility`*, *`RespApplication`*, *`SecurityGroup`*, *`NLSLanguage`*, and *`Org_Id`*.

> **Note:** The user information is defined by the `wsseUsername` property passed within the security headers. Detailed instructions on how to pass the security headers along with the SOAP request, see Passing Values to Security Headers, page 7-7.

The expected values for these elements are described in the following table:

Header Variables and Expected Values for Concurrent Program Interface Type

Element Name	Expected Value
`Responsibility`	`responsibility_key` (such as "SYSTEM_ADMINISTRATOR")
`RespApplication`	Application Short Name (such as "FND")
`SecurityGroup`	Security Group Key (such as "STANDARD")
`NLSLanguage`	NLS Language (such as "AMERICAN")
`Org_Id`	Org Id (such as "202")

Note: NLS Language and Org_Id are optional values to be passed.

- If the NLS Language element is specified, SOAP requests can be consumed in the language passed. All corresponding SOAP responses and error messages can also be returned in the same language. If no language is identified, then the default language of the user will be used.

- If a service execution is dependent on any particular organization, then you must pass the Org_Id element of that SOAP request.

The context information can be specified by configuring an Assign activity before the Invoke activity in the BPEL PM.

BPEL Process Creation Flow

Based on the scenario, the following design-time tasks are discussed in this chapter:

1. Create a New BPEL Project, page 7-5

   Use this step to create a new BPEL project called `ShipNotice.bpel`. This automatically creates two dummy activities - Receive and Reply - to receive input from a third party application and to reply output of the BPEL process back to the request application.

2. Create a Partner Link, page 7-5

   Use this step to create a partner link for the Departure Shipment Notice Outbound `Shipment_Notice` concurrent service.

3. Add a Partner Link for File Adapter, page 7-8

   This is to synchronous read sales order details received from the trading partner.

4. Add Invoke Activities, page 7-12

   Use this step to create two Invoke activities in order to:

   1. Point to the File Adapter - Synchronous Read operation to read the order details from the Assign activity.

   2. Point to the `Shipment_Notice` web service to create the shipment notice with header and line details.

5. Add Assign Activities, page 7-15

   Use this step to create three Assign activities in order to:

   1. To pass the SOAHeader variables for the invocation of the DSNO concurrent program service.

2. To pass the order details from the output of the Synchronous Read - File Adapter service to the input of the DSNO creation.

3. To set the SOAP response to output.

For general information and basic concept of a BPEL process, see Understanding BPEL Business Processes, page D-1 and *Oracle BPEL Process Manager Developer's Guide* for details.

Creating a New BPEL Project

Use this step to create a new BPEL project that will contain various BPEL process activities.

To create a new BPEL project:

1. Open JDeveloper BPEL Designer.

2. From the **File** menu, select **New**. The New Gallery dialog box appears.

3. Select **All Items** from the **Filter By** box. This produces a list of available categories.

4. Expand the **General** node and then select **Projects**.

5. Select **BPEL Process Project** from the **Items** group.

6. Click **OK**. The BPEL Process Project dialog box appears.

7. In the **Name** field, enter a descriptive name for example `ShipNotice`.

8. From the Template list, select **Synchronous BPEL Process** and then select **Use Default Project Settings**.

9. Use the default input and output schema elements in the Input/Output Elements dialog box.

10. Click **Finish**.

    A new synchronous BPEL process is created with the Receive and Reply activities. The required source files including bpel.xml, using the name you specified (for example, `ShipNotice.bpel`) are also generated.

Creating a Partner Link for the Web Service

Use this step to create a Partner Link called `Shipment_Notice` for the web service exposed through `WSHDSNO` concurrent program.

To create a partner link for Shipment_Notice Web service:

1. In JDeveloper BPEL Designer, drag and drop the **PartnerLink** service from the Component Palette into the Partner Link border area of the process diagram. The Service Name dialog box appears.

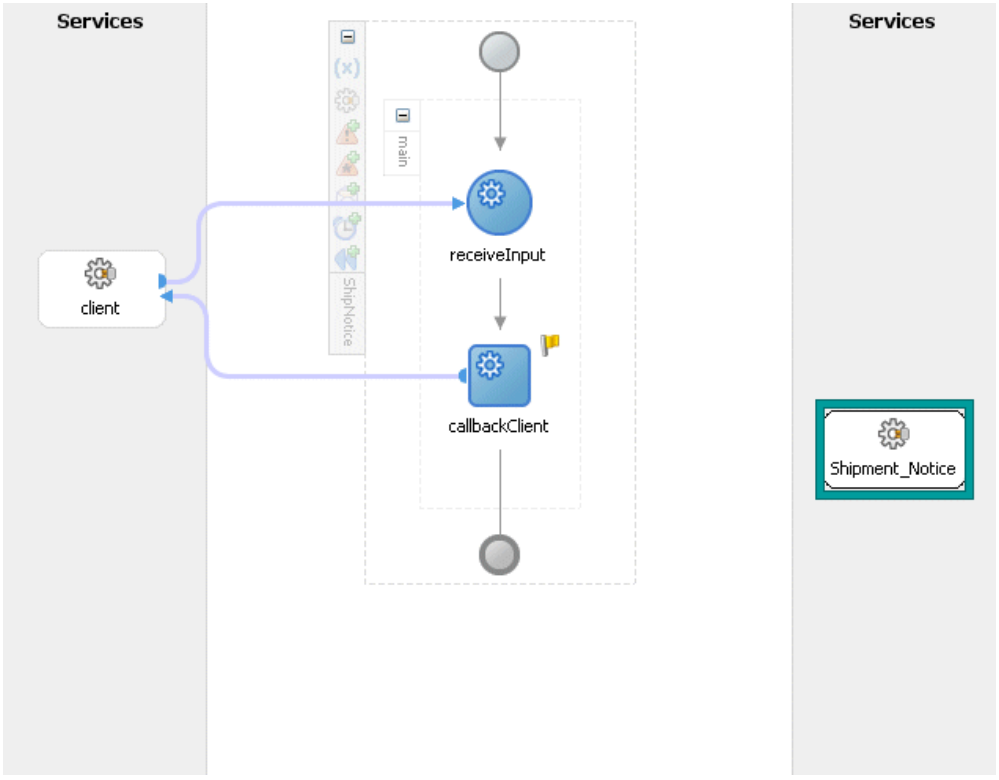
2. Copy the WSDL URL corresponding to the Departure Shipment Notice Outbound `WSHDSNO` service that you recorded earlier in the WSDL File field. Click **OK**.

3. A Partner Link Type message dialog box appears asking whether you want the system to create a new WSDL file that will by default create partner link types for you.

   Click **Yes** to have the Partner Name value populated automatically.

   The partner link is created with the required WSDL settings, and is represented in the BPEL project by a new icon in the border area of the process diagram.

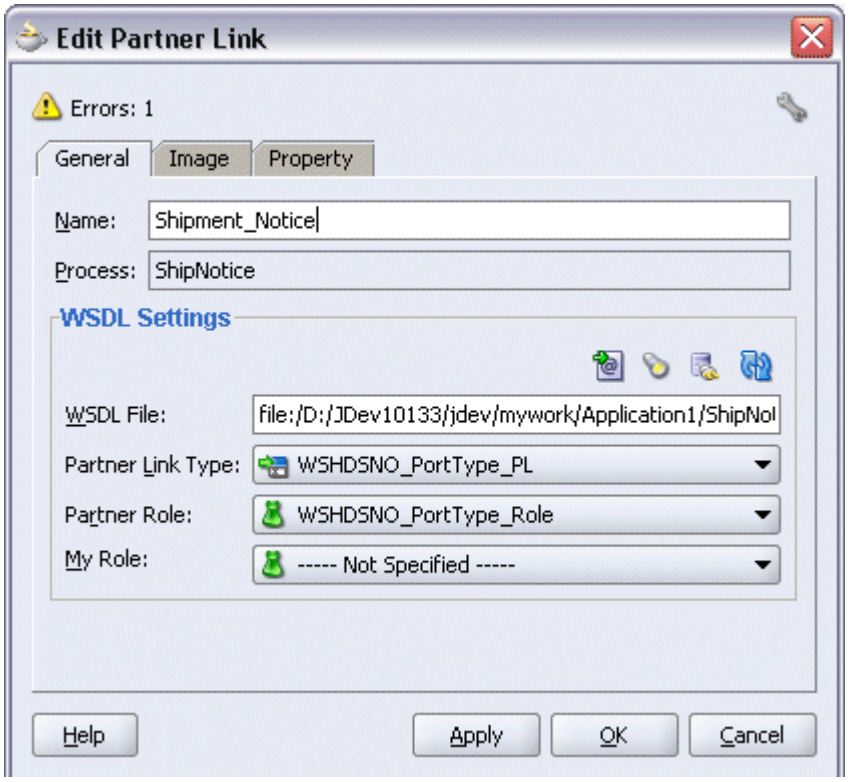
BPEL Process Diagram with the Web Service Partner Link



4. You can optionally change the default partner link name by double-clicking the icon to open the Edit Partner Link window. For example, change it from `WSHDSNO` to `Shipment_Notice`.

   Select the Partner Role value from the drop-down list. Click **Apply**.

Edit Partner Link Dialog



5. **Passing Values to Security Headers**

Select the Property tab and click the **Create Property** icon to select the following properties from the property name drop-down list in order to pass the security headers along with the SOAP request:

- wsseUsername

  Specify the username to be passed in the Property Value box.

- wssePassword

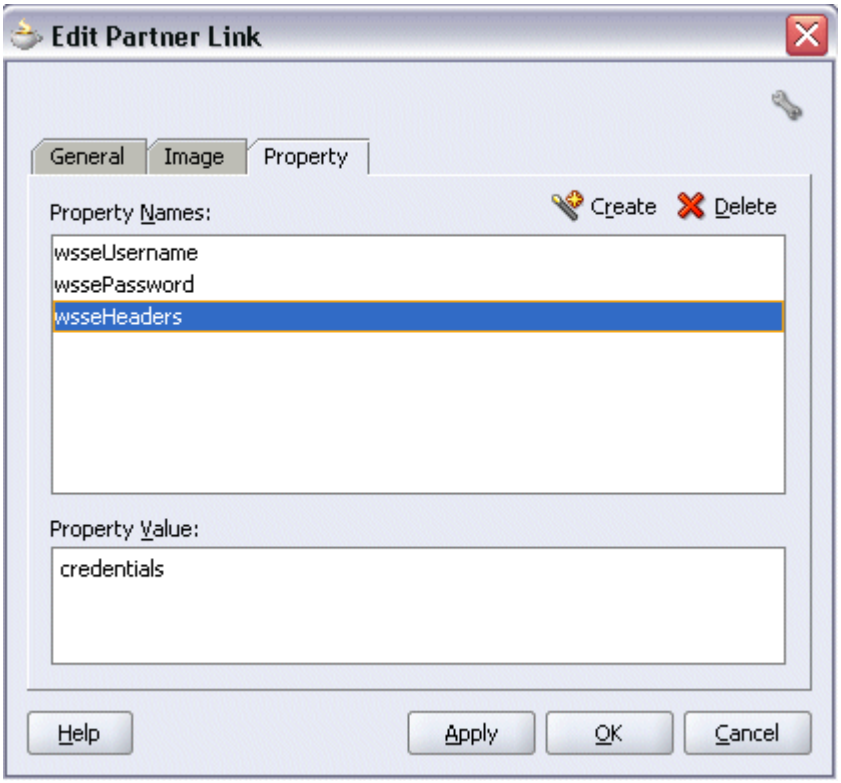
  Specify the corresponding password for the username to be passed in the Property Value box.

- wsseHeaders

  Enter `credentials` as the property value.

Click **Apply** to save the selected property values.

Edit Partner Link Dialog: Property Tab



6. Click **OK** to complete the partner link configuration.

Adding a Partner Link for File Adapter

Use this step to configure a BPEL process by synchronously reading a sales order to obtain the order details.

To add a Partner Link for File Adapter to read order details:

1. In JDeveloper BPEL Designer, drag and drop the **File Adapter** service from the **Adapter Service** section of the Component Palette into the Partner Link area of the process diagram. The Adapter Configuration wizard welcome page appears.

2. Click **Next**. The Service Name dialog box appears.

3. Enter a name for the file adapter service, for example `ReadOrder`. You can add an optional description of the service.

4. Click **Next** and the Operation dialog box appears.

Operation Dialog

5. Specify the operation type, for example **Synchronous Read File**. This automatically populates the **Operation Name** field.

   Click **Next** to access the File Directories dialog box.

6. Select **Physical Path** radio button and enter the input payload file directory information. For example, enter `/usr/tmp/` as the directory name.

   Uncheck the **Delete Files after successful retrieval** check box. Click **Next** to open the File Name dialog box.

7. Enter the name of the file for the synchronous read file operation. For example, enter 'order_data.xml'. Click **Next**. The Messages dialog box appears.

8. Select **Browse** for schema file in Schema Location. The Type Chooser window is displayed.

   1. Click **Import Schema Files** button on the top right corner of the Type Chooser window.

   2. Enter the schema location for the service. Such as `http://<myhost>:<port>/webservices/SOAProvider/concurrentprogram/wshdsno/APPS_ISG_CP_REQUEST_CP_SUBMIT.xsd`.

Schema location for your service can be found from the service WSDL URL (for example, `http://<myhost>:<port>/webservices/SOAProvider/concurrentprogram/wshdsno/?wsdl`).

3. Select the **Add to Project** check box and click **OK**.

4. Click **OK** for Import schema prompt.

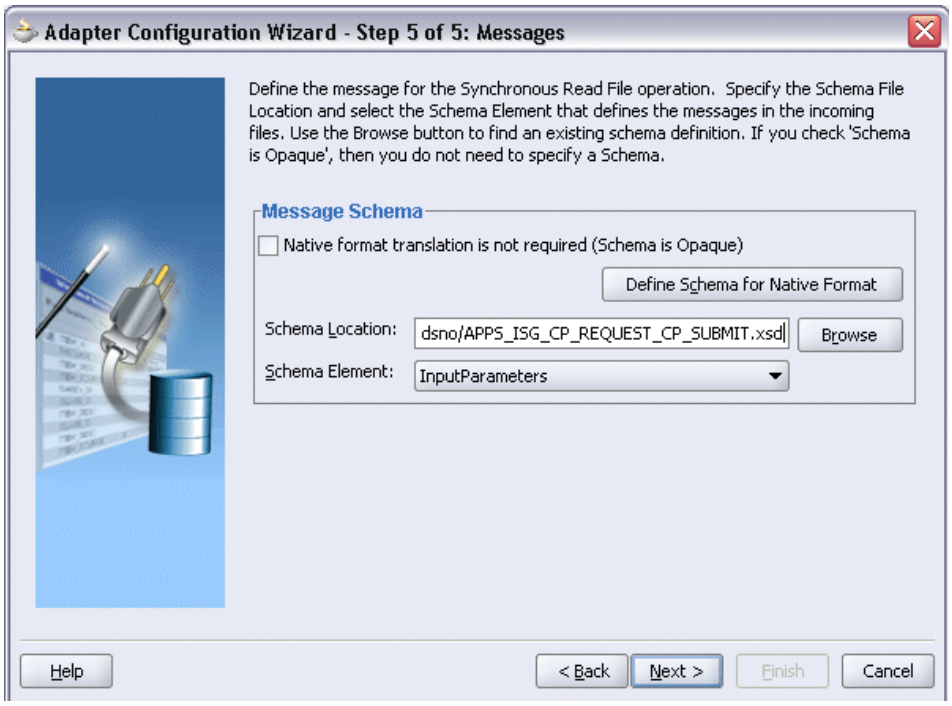
   The Imported Schemas folder is automatically added to the Type Chooser window.

5. Select schema element by expanding the Imported Schemas folder > APPS_ISG_CP_REQUEST_CP_SUBMIT.xsd > InputParameters.

   Click **OK**.

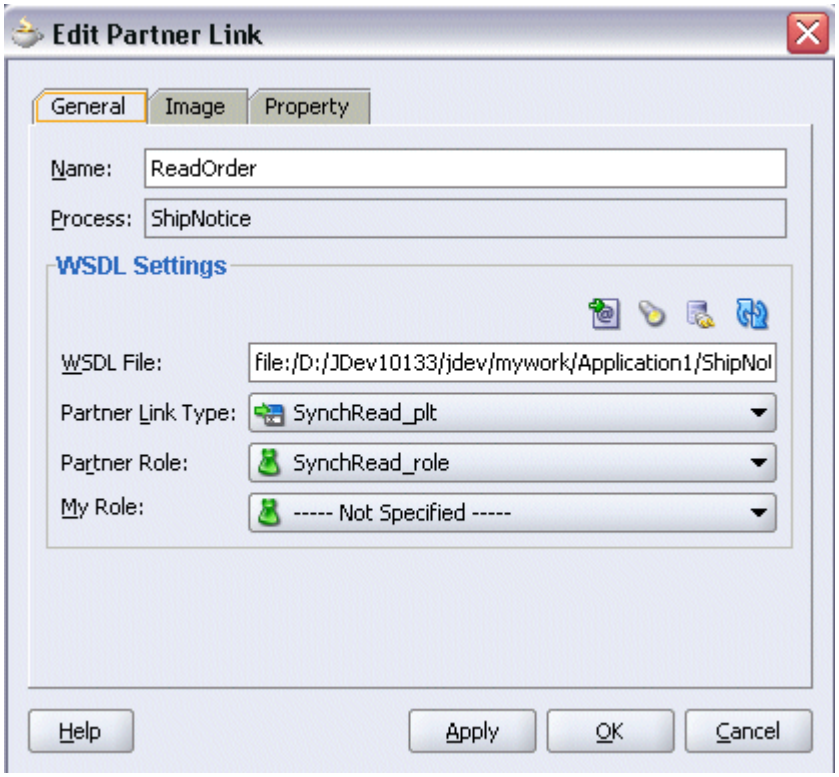
   The selected schema location and element values are displayed.

Messages Dialog with Selected Message Schema and Element



9. Click **Next** and then **Finish**. The wizard generates the WSDL file corresponding to the partner link. The main Create Partner Link dialog box appears, specifying the new WSDL file `ReadOrder.wsdl`.

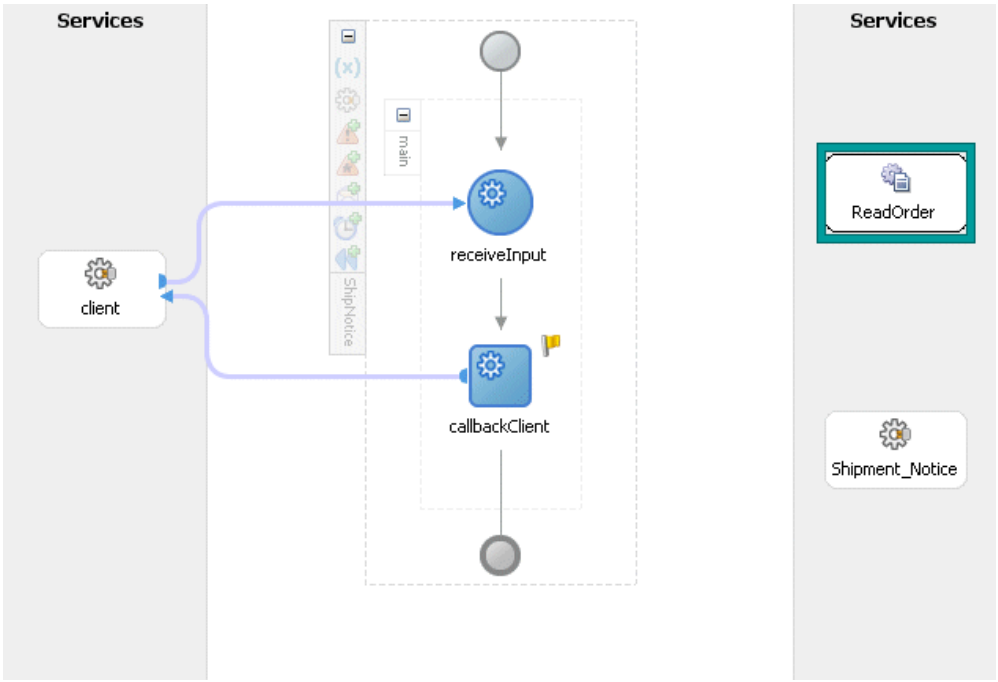
Edit Partner Link Dialog with Partner Link Configuration



Click **Apply** and then **OK** to complete the configuration and create the partner link with the required WSDL settings for the File Adapter Service.

The `ReadOrder` Partner Link appears in the BPEL process diagram.

BPEL Process Dialog with File Adapter Partner Link



Adding Invoke Activities

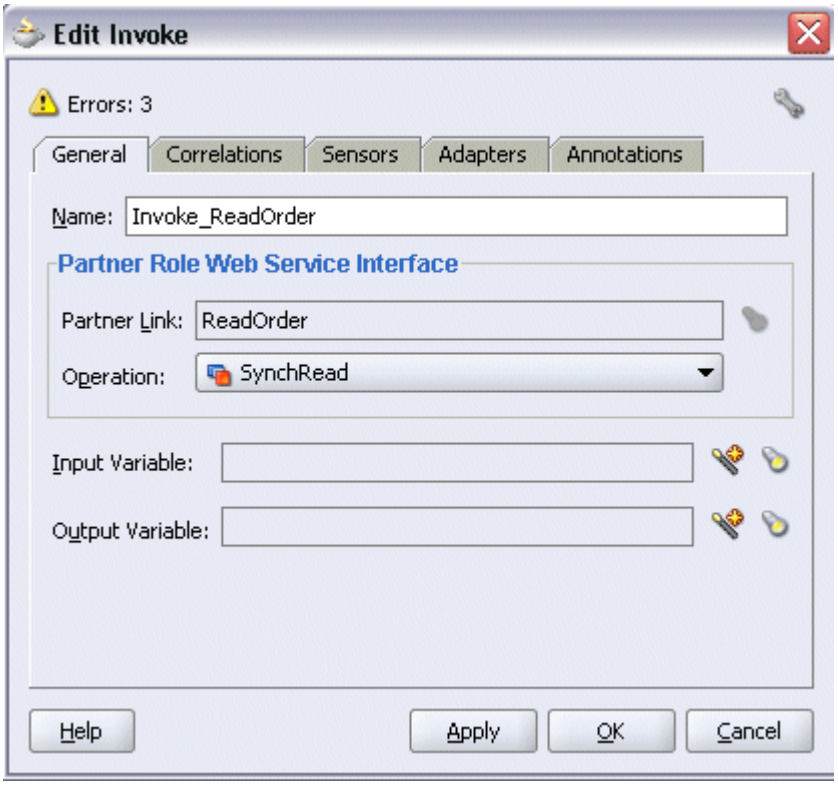
This step is to configure two Invoke activities:

- Read order details that is passed from the first Assign activity through the `ReadOrder` partner link for File Adapter.
- Send the order header and line details received from the Assign activities to generate an outbound shipment notice (DSNO) by using the `Shipment_Notice` partner link.

To add an Invoke activity for ReadOrder Partner Link:

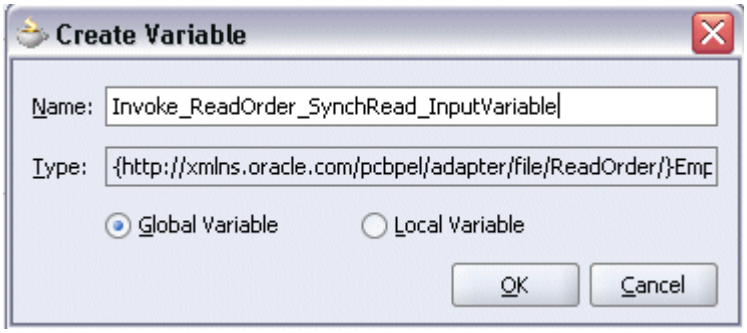
1. In Oracle JDeveloper BPEL Designer, drag and drop the **Invoke** activity from the Component Palette into the Activity box of the process diagram, between the **Receive** and **Reply** activities.
2. Link the Invoke activity to the `ReadOrder` service. The Invoke activity will send order data to the partner link. The Edit Invoke dialog box appears.

Edit Invoke Dialog



3. Enter a name for the Invoke activity, and then click the **Create** icon next to the **Input Variable** field to create a new variable. The Create Variable dialog box appears.

Create Variable Dialog



4. Select **Global Variable** and then enter a name for the variable. You can also accept the default name. Click **OK**.

5. Click the **Create** icon next to the **Output Variable** field. Select **Global Variable** and

then enter a name for the variable. You can also accept the default name. Click **OK**.

6. Click **Apply** and then **OK** in the Edit Invoke dialog box to finish configuring the Invoke activity.

   The Invoke activity appears in the process diagram.

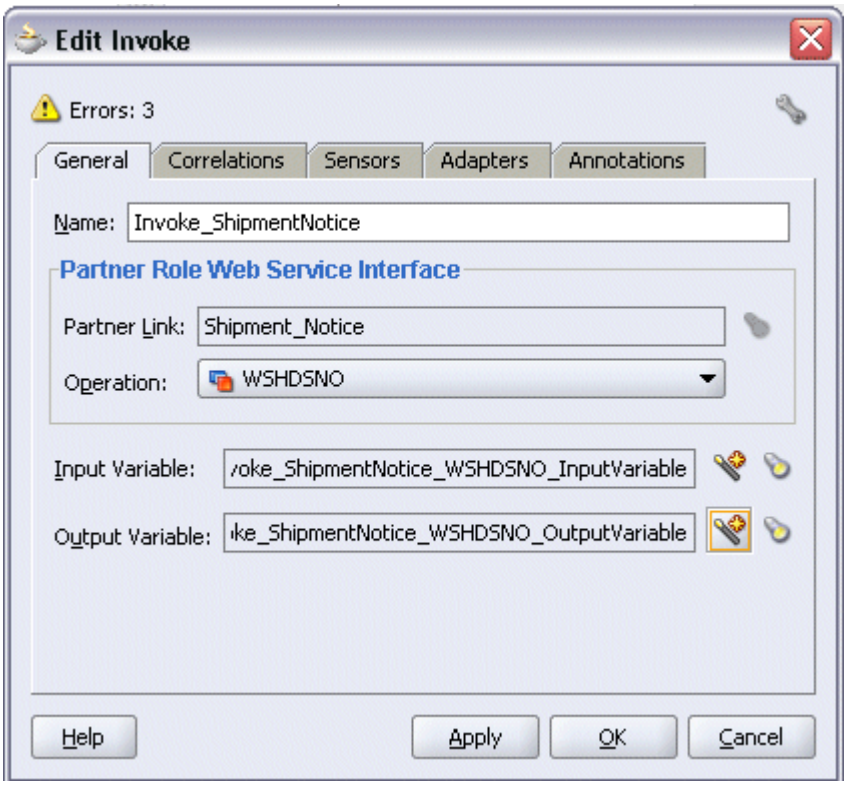
To add the second Invoke activity for Shipment_Notice Partner Link:

1. In Oracle JDeveloper BPEL Designer, drag and drop the **Invoke** activity from the Component Palette into the Activity box of the process diagram, after the **Invoke** and **Reply** activities.

2. Link the Invoke activity to the `Shipment_Notice` service. The Invoke activity will send event data to the partner link. The Edit Invoke dialog box appears.

3. Enter a name for the Invoke activity such as 'Invoke_ShipmentNotice'. Select input and output global variables as described in the first Invoke activity creation procedure.

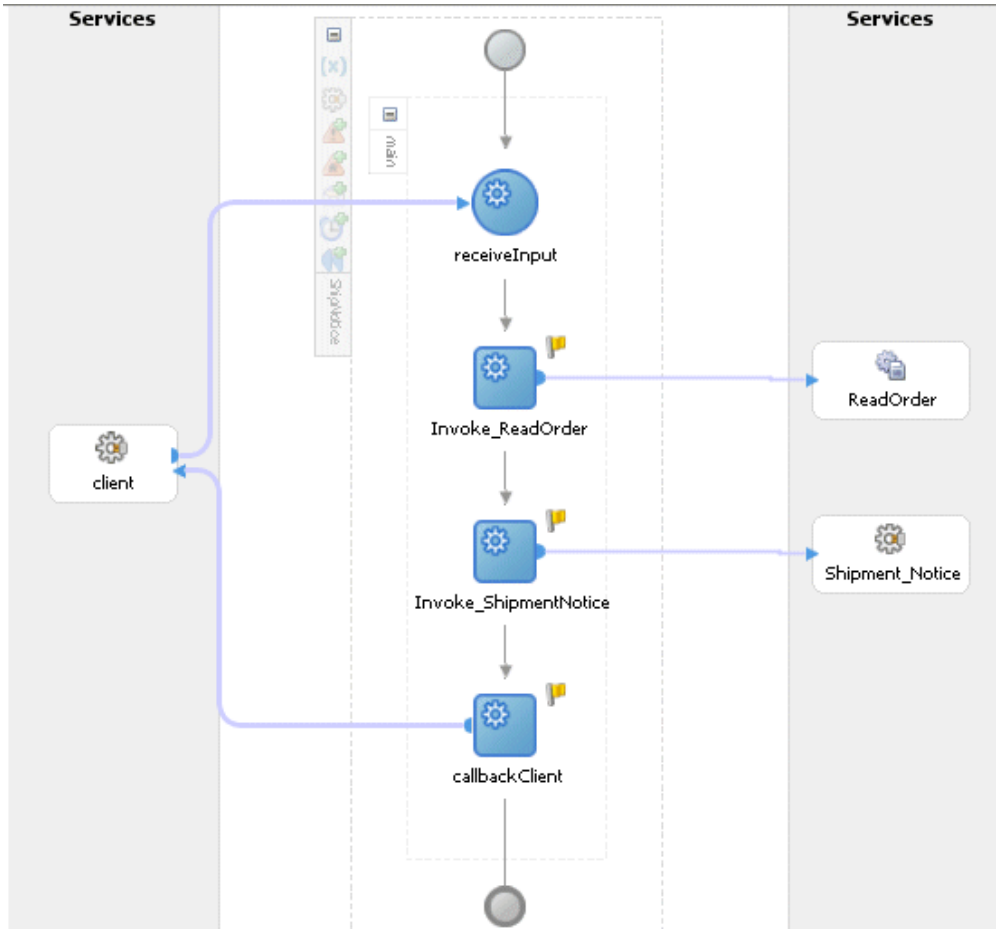
Edit Invoke Dialog



Click **Apply** and then **OK** in the Edit Invoke dialog box to finish configuring the Invoke activity.

The second Invoke activity appears in the process diagram.

BPEL Process Dialog with Invoke Activities



Adding Assign Activities

This step is to configure three Assign activities:

1. To pass the application context for SOAHeader in the invocation of the DSNO concurrent program service.

2. To pass the order details from the output of the Synchronous Read - File Adapter service to the input of the DSNO creation through the Invoke_ShipmentNotice Invoke activity.

3. To set the SOAP response to output.

Assigning SOAHeader Parameters:

To add the first Assign activity to pass SOAHeader variables used in the invocation of

the DSNO concurrent program service:

1. Add the first Assign activity by dragging and dropping the **Assign** activity from the Component Palette into the Activity box of the process diagram, between the two **Invoke** activities.

2. Enter 'SOAHeader' as the Assign name in the Edit Assign dialog box. Click **OK**.

3. Enter the first pair of parameters by selecting **Copy Operation** from the Create drop-down list with the following values:

   - In the From navigation tree, select type Expression and then enter 'ORDER_MGMT_SUPER_USER' in the Expression box.

   - In the To navigation tree, select type Variable. Navigate to **Variable > Process > Variables > Invoke_Shipment_Notice_WSHDSNO_InputVariable >header > ns2:SOAHeader** and select **ns2:Responsibility**. The XPath field should contain your selected entry.

   - Click **OK**.

4. Enter the second pair of parameters by selecting **Copy Operation** from the Create drop-down list with the following values:

   - In the From navigation tree, select type Expression and then enter 'ONT' in the Expression box.

   - In the To navigation tree, select type Variable. Navigate to **Variable > Process > Variables > Invoke_Shipment_Notice_WSHDSNO_InputVariable >header > ns2:SOAHeader** and select **ns2:RespApplication**. The XPath field should contain your selected entry.

   - Click **OK**.

5. Enter the third pair of parameters by selecting **Copy Operation** from the Create drop-down list with the following values:

   - In the From navigation tree, select type Expression and then enter 'STANDARD' in the Expression box.

   - In the To navigation tree, select type Variable. Navigate to **Variable > Process > Variables > Invoke_Shipment_Notice_WSHDSNO_InputVariable >header > ns2:SOAHeader** and select **ns2:SecurityGroup**. The XPath field should contain your selected entry.

   - Click **OK**.

6. Enter the fourth pair of parameters by selecting **Copy Operation** from the Create

drop-down list with the following values:

- In the From navigation tree, select type Expression and then enter 'AMERICAN' in the Expression box.
- In the To navigation tree, select type Variable. Navigate to **Variable > Process > Variables > Invoke_Shipment_Notice_WSHDSNO_InputVariable >header > ns2:SOAHeader** and select **ns2:NLSLanguage**. The XPath field should contain your selected entry.
- Click **OK**.

7. Enter the fifth pair of parameters by selecting **Copy Operation** from the Create drop-down list with the following values:
   - In the From navigation tree, select type Expression and then enter '202' in the Expression box.
   - In the To navigation tree, select type Variable. Navigate to **Variable > Process > Variables > Invoke_Shipment_Notice_WSHDSNO_InputVariable >header > ns2:SOAHeader** and select **ns2:Org_Id**. The XPath field should contain your selected entry.
   - Click **OK**.
8. The Edit Assign dialog box appears.

   Click **Apply** and then **OK** to complete the configuration of the Assign activity.

To add the second Assign activity to set order details to the Invoke_ShipmentNotice Invoke activity:

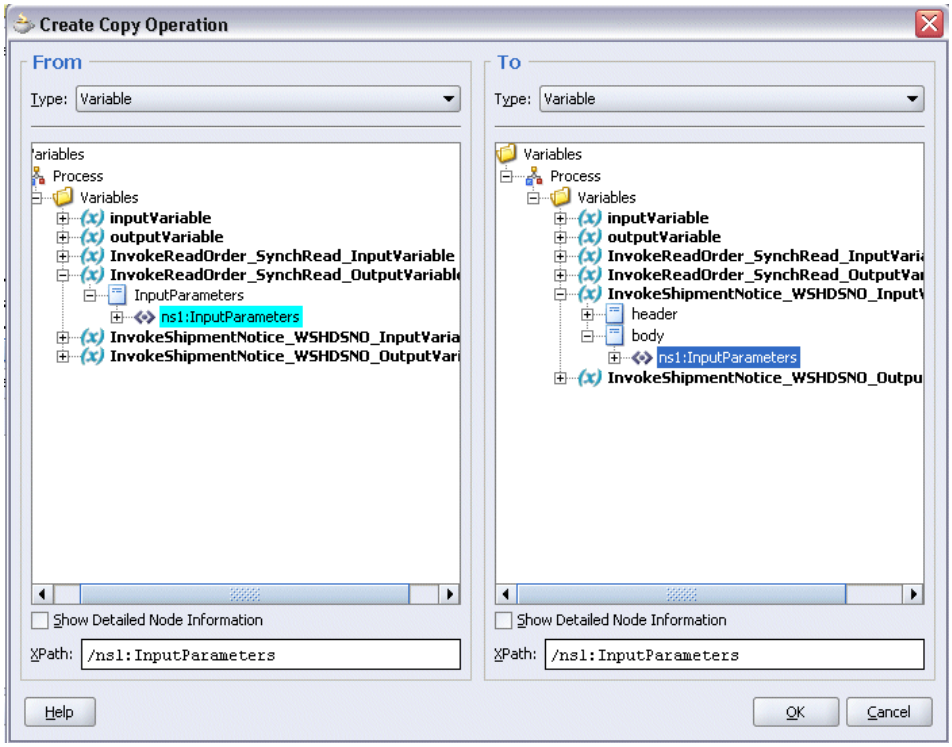
1. In JDeveloper BPEL Designer, drag and drop the **Assign** activity from the Component Palette into the Activity box of the process diagram, between **Assign** and **Invoke** activities.
2. Double-click the **Assign** activity to access the Edit Assign dialog box.
3. Click the General tab to enter the name for the Assign activity, such as 'SetOrderDetails'.
4. On the Copy Operation tab, click **Create** and then select **Copy Operation** from the menu. The Create Copy Operation window appears.
5. Enter the first pair of parameters:
   - In the From navigation tree, select type elect type Variable. Navigate to **Variable > Process > Variables >**

InvokeReadOrder_SynchRead_OutputVariable > InputParameters and select **ns1:InputParametersr**.

The XPath field should contain your selected entry.

- In the To navigation tree, select type Variable. Navigate to **Variable > Process > Variables > InvokeShipmentNotice_WSHDSNO_InputVariable > body** and select **ns1:InputParameters**. The XPath field should contain your selected entry.
- Click **OK**.

Create Copy Operation Dialog for Assigning Parameters



6. The Edit Assign dialog box appears.

   Click **Apply** and then **OK** to complete the configuration of the Assign activity.

To add the third Assign activity to set SOAP response to output:

1. Add the third Assign activity by dragging and dropping the **Assign** activity from the Component Palette into the Activity box of the process diagram, between the **Invoke** and the **Reply** activities.

2. Repeat Step 2 to Step 4 described in creating the first Assign activity to add the third Assign activity called 'SetCPdetails'.

3. Enter the following information:

   - In the From navigation tree, select type Variable. Navigate to **Variable > Process > Variables > InvokeShipmentNotice_WSHDSNO_OutputVariable** and select **body**.

   - In the To navigation tree, select type Variable. Navigate to **Variable > Process > Variables > OutputVariable** and select **payload**.

   - Click **OK**.

4. Click **Apply** and then **OK** to complete the configuration of the Assign activity.

Deploying and Testing the BPEL Process at Run Time

After creating a BPEL process using the WSDL URL generated from the concurrent program interface definition, you can deploy it to a BPEL server if needed. To ensure that this process is modified or orchestrated appropriately, you can also manually test the BPEL process by initiating the business process contained in the BPEL process to test the interface integration.

Prerequisites

Before deploying the BPEL process using Oracle JDeveloper, you must ensure that you have established the connectivity between the deign-time environment and the run-time servers including the application server and the integration server.

How to configure the necessary server connection, see Configuring Server Connection, page B-1.

To validate your BPEL process, perform the following run-time tasks:

1. Deploy the BPEL process, page 7-19

   Once you deploy the process to a BPEL server, it becomes available so that you can run the process manually to test it for validation.

2. Test the BPEL process, page 7-20

   After deploying a BPEL process, you can manage the process from the BPEL console to validate the interface integration contained in your BPEL process.

Deploying the BPEL Process

You must deploy the BPEL process (`ShipNotice.bpel`) that you created earlier before you can run it.

To deploy the BPEL process:

1. In the Applications Navigator of Oracle JDeveloper BPEL Designer, select the

ShipNotice project.

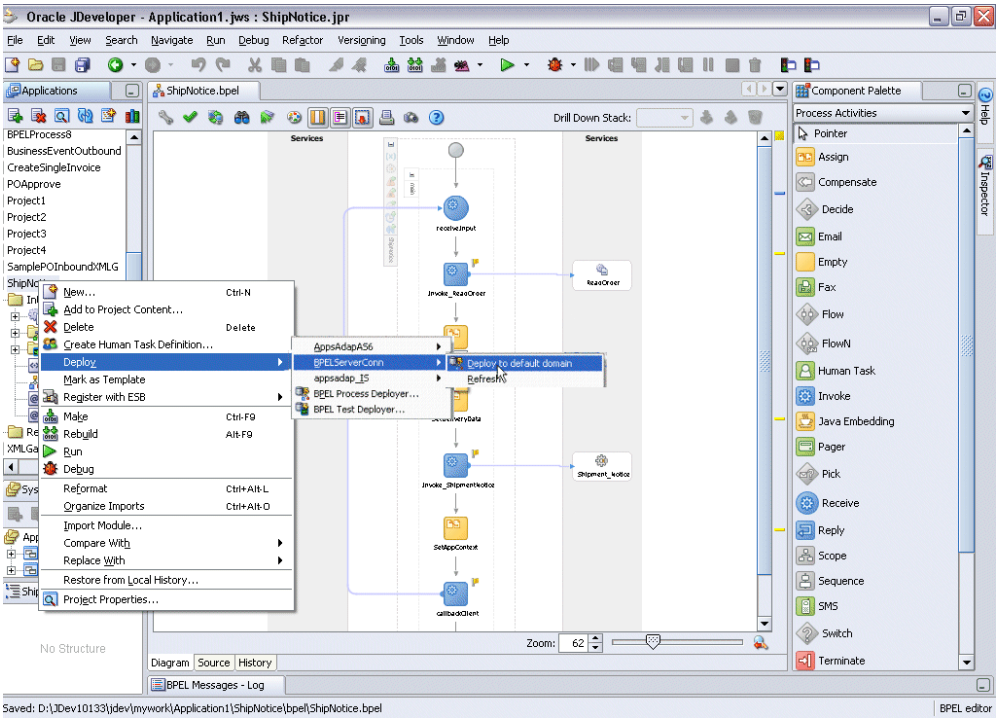
2. Right-click the project and click **Make** action from the menu.

   Look for any compilation error messages in Message Log.

   Right-click the project and select **Deploy >*Integration Server Connection name* > Deploy to Default Domain** action from the menu.

   For example, you can select **Deploy > BPELServerConn > Deploy to Default Domain** to deploy the process if you have the BPEL Process Manager setup appropriately.

Oracle JDeveloper for Deploying the BPEL Process



3. Look for 'Build successful' message in Apache Ant – Log. The BPEL project is compiled and successfully deployed.

Testing the BPEL Process

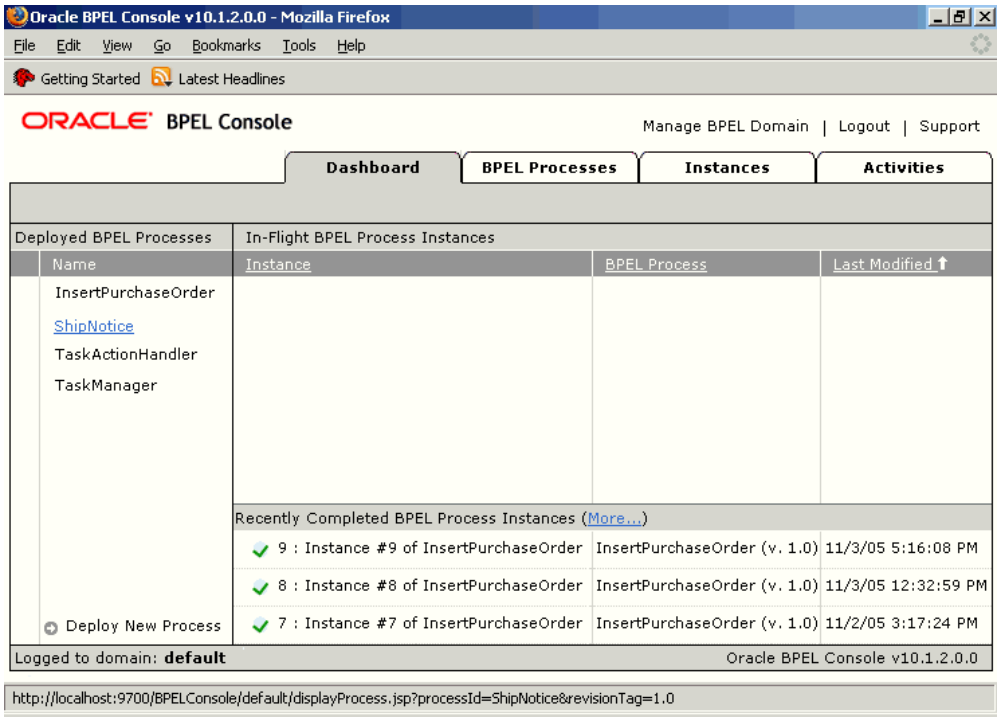
To validate whether the BPEL process that you created works or not, you need to manually initiate the process after it has been successfully deployed to the BPEL server. Therefore, the validation starts with the BPEL console to ensure that you can find the deployed BPEL process listed in the console. Then, you can log on to Oracle E-Business Suite to manually initiate the processes and to confirm that the departure shipment notice outbound (DSNO) is generated in the XML file.

To test the BPEL process:

1. Log in to Oracle Application Server 10g BPEL Console (`http://<soaSuiteServerHostName>:<port>/BPELConsole`). The BPEL Console login page appears.

2. Enter the username and password and click **Login**. The Oracle Enterprise Manager 10g BPEL Console appears with a list of deployed BPEL processes.

Oracle Enterprise Manager BPEL Console to View the Deployed Service



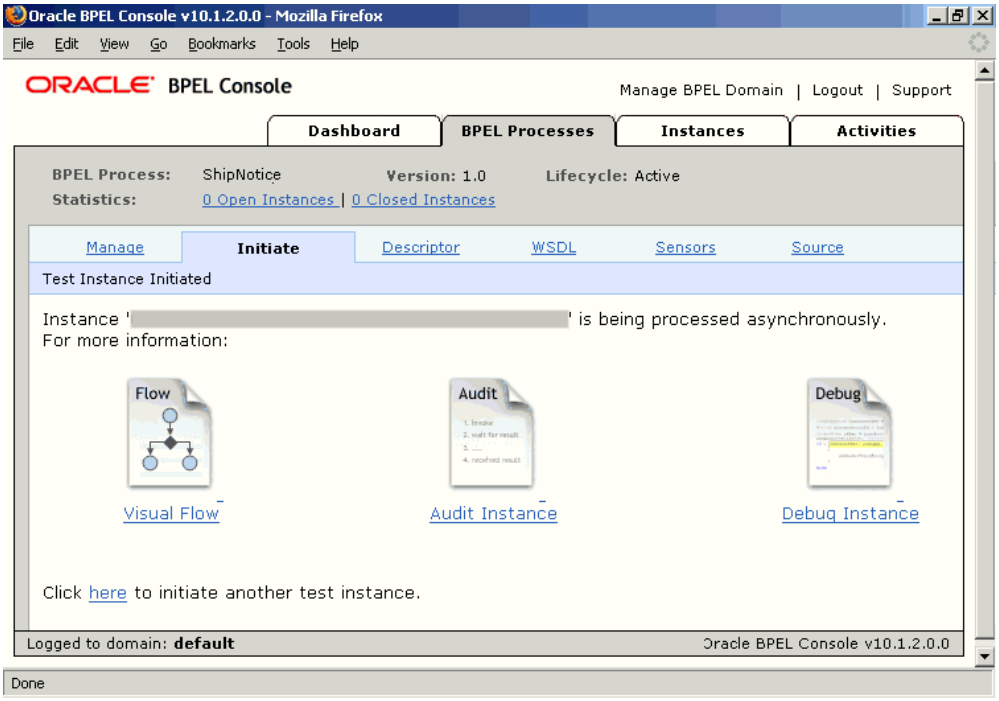
3. In the BPEL Console, confirm that `ShipNotice` has been deployed.

4. Click the ShipNotice link to open the Initiate tab.

5. Enter the payload input field and click **Post XML Message** to initiate the process.

6. The BPEL process is now initiated. You can check the process flow by clicking the **Visual Flow** icon.

BPEL Console: BPEL Processes Tab: Initiate Subtab



7. Double-click the Invoke_ShipmentNotice icon from the process flow chart and click **View XML document** link to open the XML file. This file records the Request ID that is returned for the transaction.

Verifying Records in Oracle E-Business Suite

Before verifying the records in Oracle E-Business Suite, you must first ensure that the concurrent request is completed successfully.

1. Log in to Oracle E-Business Suite with the System Administrator responsibility. Select **View > Requests** to open the Find Requests window.

2. Search for the concurrent request by entering the Request Id that you got from the audit trail and then click **Find**.

3. The request details page is displayed. You can check the Phase and Status of the request to see if the Status of the request is `Complete`.

Once the concurrent request is completed successfully, you can validate it in Oracle E-Business Suite.

Since DSNO (departure shipment notice outbound) is an outbound XML message, relevant XML Gateway setup tasks must be configured appropriately in order for the shipment notice to be delivered to the right recipient.

See *Oracle XML Gateway User's Guide* for details.

You can validate if the ship-to address, purchase order, and requested ship date addressed in the DSNO XML file are the same in your sales order.

8

Using Open Interface REST Services

Overview

Similar to concurrent program REST services, open interface tables and views can now be deployed as REST services. This allows you to leverage the deployed open interface REST services to update or import data directly into Oracle E-Business Suite base tables or make the base table data available for selection.

> **Note:** Open interface tables and views can be available as REST services only. The custom open interfaces are not supported in this release.

To locate an open interface table or view, in the Search page, select "Open Interface" or "Interface View" as the interface type and enter additional search criteria if desired to retrieve your desired interface. An integration administrator can then deploy it as a REST service with supported HTTP methods.

Service Invocation Examples

To better understand how to use open interface REST services to modify Oracle E-Business Suite base tables, this chapter includes examples of using HTTP methods to insert, update, fetch, and delete data through the use of an open interface table REST service.

Using an Open Interface Table REST Service

REST Service Invocation Scenario

An open interface table 'AR Autoinvoice' (`RAXMTR`) is used in this example to explain how you can test and use an open interface table REST service.

In this example, a few HTTP requests are received to insert, retrieve, and remove invoice data from Oracle Receivables. The `RA_INTERFACE_LINES_ALL` REST operation contained in the 'AR Autoinvoice' interface is invoked to execute these requests.

After each successful service invocation, the client will receive a REST response message to indicate the invoice line data has been successful inserted, updated, retrieved, or removed from Oracle Receivables.

Invoking an Open Interface REST Service

Based on the REST service invocation scenario, this chapter includes the following topics:

1. Deploying a REST Service, page 8-2
2. Creating a Security Grant, page 8-3
3. Recording Resource Information from Deployed WADL, page 8-4
4. Invoking a REST Service Using Command Lines, page 8-4

Deploying a REST Service

Use the following steps to deploy the open interface table 'AR Autoinvoice' (RAXMTR):

1. Log in to Oracle E-Business Suite as a user who has the Integration Administrator role.

   Select the Integrated SOA Gateway responsibility and the Integration Repository link from the navigation menu.

2. In the Integration Repository tab, click **Search** to access the main Search page.

3. Enter the following key search values as the search criteria:

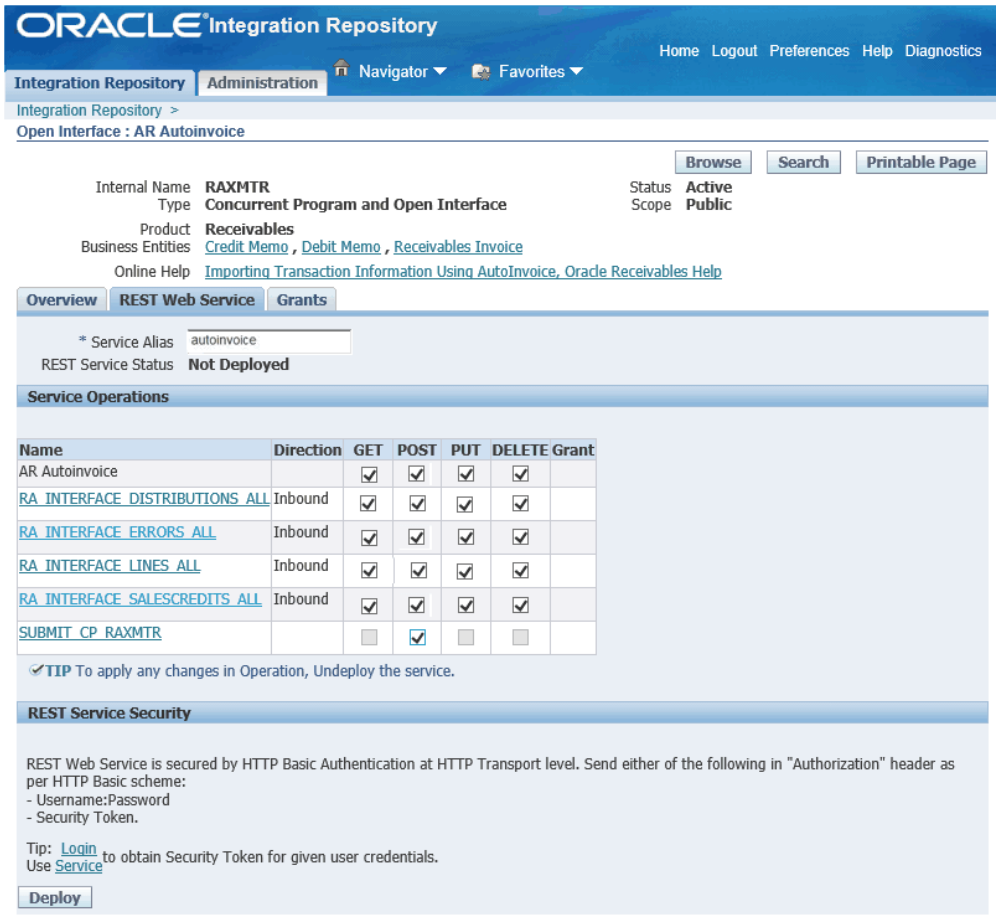
   - Internal Name: RAXMTR
   - Interface Type: Open Interface

4. Click **Go** to execute the search.

   Click the 'AR Autoinvoice' interface name link to open the interface details page.

5. In the REST Web Service tab, enter the following information:

Open Interface Table Interface Details Page with REST Web Service Tab



- Service Alias: autoinvoice

  The alias will be displayed as the service endpoint in the WADL and schema for the selected method or operation.

- Select Desired Service Operations

  In the first row 'AR Autoinvoice', select the four HTTP method check boxes (GET, POST, PUT, and DELETE). The rest of the open interface table names shown as the method names listed in the table are all automatically selected.

6. Click **Deploy** to deploy the service to an Oracle E-Business Suite WebLogic environment.

Creating a Security Grant

After deploying the 'AR Autoinvoice' as a REST service, the integration administrator can create a security grant to authorize the service or method access privileges to a

specific user, a user group, or all users.

Use the following steps to create a security grant:

1. Log in to Oracle E-Business Suite as a user who has the Integration Administrator role. Select the Integrated SOA Gateway responsibility and the Integration Repository link from the navigation menu.

2. Perform a search to locate the AR Autoinvoice service the administrator just deployed earlier.

3. In the interface details page of the selected AR Autoinvoice, click the Grants tab.

4. Select the `RA_INTERFACE_LINES_ALL` method check box and then click **Create Grant**.

5. In the Create Grants page, select "All User" as the Grantee Type.

   Note: In this example, the `RA_INTERFACE_LINES_ALL` service operation is granted to all users. In actual implementation, you should define strict security rules. Create grant to a user or more appropriately to a group of users defined by roles and responsibilities.

   Click **Create Grant**. This grants the selected method access privilege to all Oracle E-Business Suite users.

Recording Resource Information from Deployed WADL

To obtain service resource information from the deployed 'AR Autoinvoce' service, an integration developer clicks the **View WADL** link in the REST Web Service tab. This displays the WADL description in a different window.

Copy or record the REST service endpoint from the WADL description. This will be used later when invoking the service.`http://<hostname>:<port>/webservices/rest/autoinvoice/RA_INTERFACE_LINES_ALL/`

Replace `autoinvoice`, the alias in this sample deployment, with the alias name you used in your deployment.

Invoking a REST Service Using Command Lines

In this example, we use a third party command line tool called `cURL` to transfer data using URL syntax. This tool is available by default on most Linux environments.

Note: You can test the REST service invocation using any REST client.

This section includes the following invocation scenarios:

Inserting Data to Oracle Receivables Using the HTTP POST Method

Creating a File with Input Payload

When an HTTP request is received to insert invoice data to Oracle Receivables, before invoking the REST service, you need to create the following xml file called `lines.xml` with required data as input payload.

```
<?xml version="1.0" encoding="UTF-8"?>
<RA_INTERFACE_LINES_ALL_Input>
 <RESTHeader>
  <Responsibility>RECEIVABLES_VISION_OPERATIONS</Responsibility>
     <RespApplication>AR</RespApplication>
     <SecurityGroupE>STANDARD</SecurityGroup>
     <NLSLanguage>AMERICAN</NLSLanguage>
     <Org_Id>204</Org_Id>
  </RESTHeader>
<Select>INTERFACE_LINE_ID, BATCH_SOURCE_NAME</Select>
<InputParameters>
<RA_INTERFACE_LINES_ALL_REC>
  <INTERFACE_LINE_ATTRIBUTE9>1</INTERFACE_LINE_ATTRIBUTE9>
    <INTERFACE_LINE_ATTRIBUTE11>1</INTERFACE_LINE_ATTRIBUTE11>
     <INTERFACE_LINE_ATTRIBUTE10>1</INTERFACE_LINE_ATTRIBUTE10>
     <ORG_ID>204</ORG_ID>
  <COMMENTS>Test REST1</COMMENTS>
     <QUANTITY>1</QUANTITY>
     <TRX_NUMBER>trxpj1</TRX_NUMBER>
     <CONVERSION_RATE>1</CONVERSION_RATE>
  <CONVERSION_DATE>11-APR-2018</CONVERSION_DATE>
     <CONVERSION_TYPE>Use</CONVERSION_TYPE>
  <ORIG_SYSTEM_SHIP_ADDRESS_ID>1030</ORIG_SYSTEM_SHIP_ADDRESS_ID>
  <ORIG_SYSTEM_SHIP_CUSTOMER_ID>1004</ORIG_SYSTEM_SHIP_CUSTOMER_ID>
  <ORIG_SYSTEM_BILL_ADDRESS_ID>1030</ORIG_SYSTEM_BILL_ADDRESS_ID>
  <ORIG_SYSTEM_BILL_CUSTOMER_ID>1004</ORIG_SYSTEM_BILL_CUSTOMER_ID>
     <TERM_ID>4</TERM_ID>
     <TERM_NAME>30 Net</TERM_NAME>
  <CUST_TRX_TYPE_ID>1</CUST_TRX_TYPE_ID>
  <CUST_TRX_TYPE_NAME>Invoice</CUST_TRX_TYPE_NAME>
     <AMOUNT>1000.00</AMOUNT>
     <CURRENCY_CODE>USD</CURRENCY_CODE>
     <DESCRIPTION>Project Invoices</DESCRIPTION>
     <LINE_TYPE>LINE</LINE_TYPE>
     <SET_OF_BOOKS_ID>1</SET_OF_BOOKS_ID>
     <BATCH_SOURCE_NAME>PROJECTS INVOICES</BATCH_SOURCE_NAME>
     <INTERFACE_LINE_ATTRIBUTE7>Line</INTERFACE_LINE_ATTRIBUTE7>
    <INTERFACE_LINE_ATTRIBUTE6>1</INTERFACE_LINE_ATTRIBUTE6>
    <INTERFACE_LINE_ATTRIBUTE5>xxxxx, Mr.
xxxxx</INTERFACE_LINE_ATTRIBUTE5>
     <INTERFACE_LINE_ATTRIBUTE4>Vision
Operations</INTERFACE_LINE_ATTRIBUTE4>
     <INTERFACE_LINE_ATTRIBUTE3>Services 01</INTERFACE_LINE_ATTRIBUTE3>
    <INTERFACE_LINE_ATTRIBUTE2>3</INTERFACE_LINE_ATTRIBUTE2>
    <INTERFACE_LINE_ATTRIBUTE1>ATZ Services</INTERFACE_LINE_ATTRIBUTE1>
     <INTERFACE_LINE_CONTEXT>PROJECTS INVOICES</INTERFACE_LINE_CONTEXT>
  </RA_INTERFACE_LINES_ALL_REC>
 </InputParameters>
</RA_INTERFACE_LINES_ALL_Input>
```

Invoking the REST Service Using Command Line Tool

Once the xml file is created, place it in the directory where you are going to test the invocation. Execute the following command to invoke the service to transfer each invoice line data into Oracle Receivables based on the input payload sent from the request:

```
curl -H 'Content-Type: application/xml' -u <usernmae>:<password> -d
@lines.xml
https://<ebshost>:
<port>/webservices/rest/<alias>/RA_INTERFACE_LINES_ALL/
```

- Replace `<username>` with an Oracle E-Business Suite user name who has the privilege of accessing the open interface table AR Autoinvoice.

- Replace <alias> with autoinvoice in this example.

Viewing the Response Message

When the REST service is successfully invoked, the following output in XML format appears:

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<OutputParameters xmlns="http://xmlns.oracle.
com/apps/ar/rest/autoinvoice/ra_interface_lines_all">
 <Summary>
 <SuccessCount>1</SuccessCount>
 <ErrorCount>0</ErrorCount>
 </Summary>
 <Result>
  <Output>
    <Index>0</Index>
    <Status>SUCCESS</Status>
   <Message></Message>
  <RA_INTERFACE_LINES_ALL_REC>
  <INTERFACE_LINE_ID/>
  <BATCH_SOURCE_NAME>PROJECTS INVOICES</BATCH_SOURCE_NAME>
   </RA_INTERFACE_LINES_ALL_REC>
   </Output>
  </Result>
</OutputParameters>
```

Updating Data in Oracle Receivables Using the HTTP PUT Method

When a request of updating data in Oracle Receivables is received, we can use the PUT method to update data based on condition.

For example, we need to update the values of QUANTITY and COMMENTS for record with TRX_NUMBER equal to 'Demo1' and ORG_ID equal to 204 in RA_INTERFACE_LINES_ALL.

Creating a File with Input Payload

In this situation, we need to first create a file linesUpdate.xml with the following content as the payload:

```
<?xml version="1.0" encoding="UTF-8"?>
<RA_INTERFACE_LINES_ALL_Input>
 <RESTHeader>
  <Responsibility>RECEIVABLES_VISION_OPERATIONS</Responsibility>
     <RespApplication>AR</RespApplication>
     <SecurityGroupE>STANDARD</SecurityGroup>
     <NLSLanguage>AMERICAN</NLSLanguage>
     <Org_Id>204</Org_Id>
  </RESTHeader>
  <Select>DESCRIPTION,TRX_NUMBER</Select>
  <InputParameters>
    <Update>
  <Filter>ORG_ID==204;TRX_NUMBER==Demo1</Filter>
     <RA_INTERFACE_LINES_ALL_REC>
        <COMMENTS>Quantity updated to 5</COMMENTS>
        <QUANTITY>5</QUANTITY>
     </RA_INTERFACE_LINES_ALL_REC>
   </Update>
 </InputParameters>
</RA_INTERFACE_LINES_ALL_Input>
```

Invoking the REST Service Using Command Line Tool

Execute the following `curl` command to update QUANTITY and COMMENTS for records with TRX_NUMBER equal to 'Demo1' and ORG_ID equals to 204 in `RA_INTERFACE_LINES_ALL`.

```
curl -H 'Content-Type: application/xml' -u <usernmae>:<password> -X PUT
-d @ linesUpdate.xml
'https://<ebshost>:
<port>/webservices/rest/<alias>/RA_INTERFACE_LINES_ALL/'
```

- Replace `<username>` with an Oracle E-Business Suite user name who has the grants to access the open interface table AR Autoinvoice.

- Replace `<alias>` with `autoinvoice` in this example.

Viewing the Response Message

When the REST service is successfully invoked, the following output in XML format appears:

```
<?xml version="1.0" encoding="UTF-8"?>
<OutputParameters xmlns="http://xmlns.oracle.
com/apps/ar/rest/autoinvoice/ra_interface_lines_all">
 <Summary>
 <SuccessCount>1</SuccessCount>
 <ErrorCount>0</ErrorCount>
 </Summary>
 <Result>
  <Output>
    <Index>1</Index>
    <Status>SUCCESS</Status>
   <Message/>
     <UpdateCount>1</UpdateCount>
     <RA_INTERFACE_LINES_ALL_REC>
      <DESCRIPTION>Project Invoices</DESCRIPTION>
      <TRX_NUMBER>Demo1</TRX_NUMBER>
     </RA_INTERFACE_LINES_ALL_REC>
   </Output>
  </Result>
</OutputParameters>
```

Fetching Data from Oracle Receivables Using the HTTP GET Method

When a request of fetching data from Oracle Receivables is received, we can use the GET method to fetch data based on condition.

Invoking the REST Service Using Command Line Tool

For example, execute the following `curl` command to fetch values of TRX_NUMBER, QUANTITY, and COMMENTS from `RA_INTERFACE_LINES_ALL` for records with TRX_NUMBER equal to 'Demo1' in ORG_ID 204.

```
curl -u <usernmae>:<password> -X GET
'https://<ebshost>:
<port>/webservices/rest/<alias>/RA_INTERFACE_LINES_ALL/?
filter=ORG_ID==204;TRX_NUMBER==Demo1&select=TRX_NUMBER,QUANTITY,
COMMENTS'
```

- Replace `<username>` with an Oracle E-Business Suite user name who has the

privilege of accessing the open interface table AR Autoinvoice.

- Replace <alias> with autoinvoice in this example.

Viewing the Response Message

When the REST service is successfully invoked, it may return the following response:

```
{
  "OutputParameters" : {
    "@xmlns" : "http://xmlns.oracle.
com/apps/ar/concurrentprogram/rest/autoinvoice/ra_interface_lines_all",
    "Summary" : {
      "Select" : "TRX_NUMBER,QUANTITY,COMMENTS",
      "Filter" : "ORG_ID==204;TRX_NUMBER==Demo1",
      "Offset" : "0",
      "Limit" : "200",
      "Sort" : null,
      "GetCount" : "1",
      "TotalCount" : "1"
    },
    "Result" : {
      "Output" : {
        "RA_INTERFACE_LINES_ALL_REC" : {
          "TRX_NUMBER" : "Demo1",
          "QUANTITY" : "1",
          "COMMENTS" : "Create"
        }
      }
    }
  }
}
```

Deleting Data in Oracle Receivables Using the HTTP DELETE Method

When a request of removing data from Oracle Receivables is received, we can use the DELETE method to delete data.

Invoking the REST Service Using Command Line Tool

For example, execute the following curl command to delete data from RA_INTERFACE_LINES_ALL with TRX_NUMBER equal to 'Demo1' in 204 ORG_ID:

```
curl -u <usernmae>:<password> -X DELETE
''https://<ebshost>:
<port>/webservices/rest/<alias>/RA_INTERFACE_LINES_ALL/?
filter=ORG_ID==204;TRX_NUMBER==Demo1'
```

- Replace <username> with an Oracle E-Business Suite user name who has the privilege of accessing the open interface table AR Autoinvoice.

- Replace <alias> with autoinvoice in this example.

Viewing the Response Message

When the REST service is successfully invoked, it may return the following response message:

```
{
  "OutputParameters" : {
    "@xmlns" : "http://xmlns.oracle.
com/apps/ar/concurrentprogram/rest/autoinvoice/ra_interface_lines_all",
    "Result" : {
      "Status" : "SUCCESS",
      "Message" : null,
      "DeleteCount" : "1"
    }
  }
}
```

9

Using Business Service Objects

Overview

A business service object, formerly known as Service Bean, is a high-level service component that allows OA Framework or BC4J (Business Components for Java) components to be deployed as web services. This type of interfaces provides access to SOA services and facilitates integration between Oracle E-Business Suite and trading partners.

Business service object interfaces can be exposed as SOAP-based and REST-based services. To better understand how to utilize business service objects, this chapter takes a SOAP service as an example to explain the service invocation. An integration repository administrator or a system integration developer can first generate a SOAP service, and then the administrator can deploy it to an application server. The developer can then orchestrate the deployed SOAP service into a meaningful BPEL process with service endpoints. This process can take the data from a business partner and then insert or update Oracle E-Business Suite if necessary.

To better understand how each individual web service can be used in inserting or updating application data, this chapter provides detailed design-time and runtime tasks to guide you through the service invocation process. For the example described in the following sections, we use Oracle JDeveloper 10.1.3.3.0 as a design-time tool to create the BPEL process and use Oracle SOA Suite BPEL server 10.1.3.3.0 for the process deployment.

This chapter includes the following topics:

Using Business Service Object SOAP Services

SOA Composite Application with BPEL Process Scenario

This example uses PurchaseOrderService `/oracle/apps/fnd/framework/toolbox/tutorial/PurchaseOrderService` business service object interface to explain the BPEL process creation.

When a purchase order approval request is received, the purchase order details is read by a File Adapter. The order data is then passed to the approvePurchaseOrder method within the PurchaseOrderService to initiate the single PO approval process. The approval information is then replied back to the requestor.

If the BPEL process is successfully executed after deployment, you should notice the purchase order status changed from `Incomplete` to `Approved`.

Prerequisites to Create a BPEL Process Using a Business Service Object Web Service

- An integration repository administrator needs to successfully generate and deploy a web service to the application server.
- An integration developer needs to locate and record the deployed WSDL URL for the BSO interface exposed as a web service.
- Header variables need to be populated for web service authorization.

Generating and Deploying WSDL URL

An integration repository administrator must first generate a SOAP service for a selected interface definition, and then deploy the service from Oracle Integration Repository to the application server.

For example, the administrator must perform the following steps before letting the integration developers use the deployed WSDL in creating a BPEL process:

1. To generate a SOAP service, locate the business service object interface definition first from the Oracle Integration Repository (such as PurchaseOrderService `/oracle/apps/fnd/framework/toolbox/tutorial/PurchaseOrderService`) and click **Generate WSDL** in the SOAP Web Service tab of the interface details page. See: Generating SOAP Web Services, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.
2. To deploy a generated SOAP service, select at least one authentication type and click **Deploy** in the SOAP Web Service tab of the interface details page to deploy the service. Once the service is successfully deployed, the selected authentication type(s) will be displayed along with 'Deployed' SOAP Service Status. For more information on securing web services with authentication types, see Managing Web Service Security, *Oracle E-Business Suite Integrated SOA Gateway Implementation*

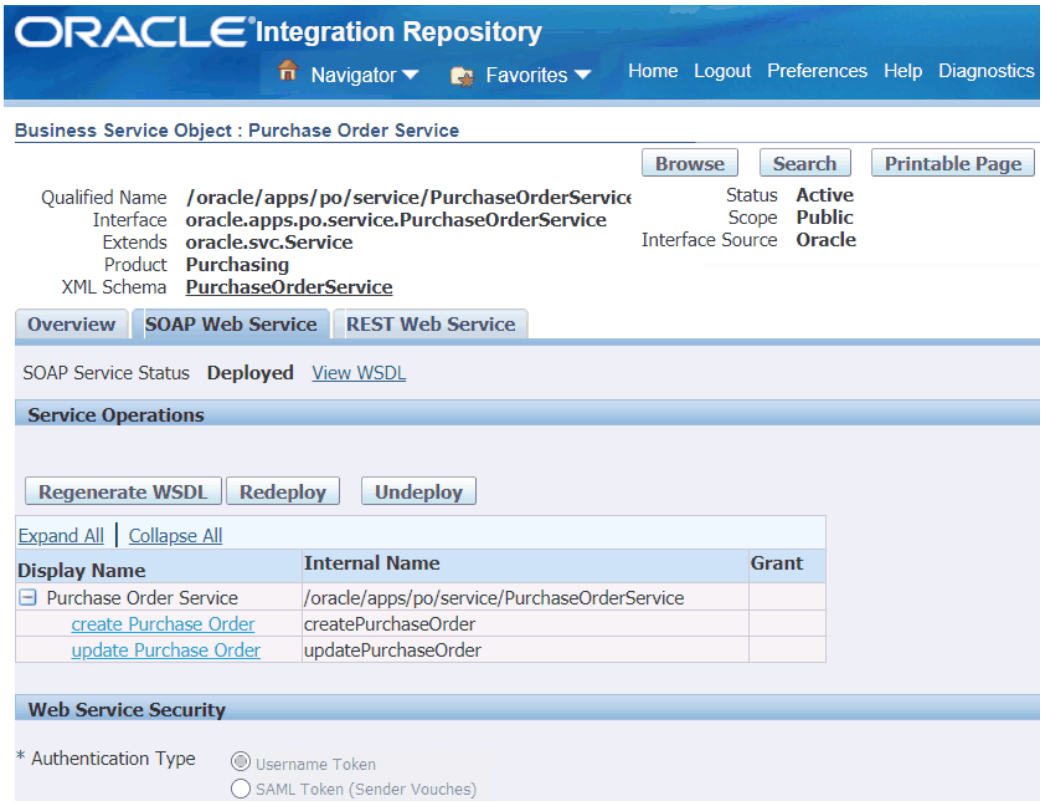
Guide.

For detailed instruction on how to deploy a web service, see Deploying and Undeploying SOAP Web Services, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

Searching and Recording WSDL URL

Apart from the required tasks performed by the administrator, an integration developer needs to log on to the system to locate and record the deployed SOAP service WSDL URL from the SOAP Web Service tab of the deployed BSO interface details page.

SOAP Web Service Tab with Deployed WSDL Link



This WSDL information will be used later in creating a partner link for the interface exposed as a SOAP service during the BPEL process creation at design time.

For information on how to search for an interface and review the interface details, see Searching and Viewing Integration Interfaces, page 2-2.

Setting Header Variables for SOAP Request

You need to populate certain variables in the BPEL process for header elements to pass values that may be required to set application context during service execution. These header elements for Business Service Object interface type are `RESPONSIBILITY_NAME, RESPONSIBILITY_APPL_NAME,`

SECURITY_GROUP_NAME, *NLS_LANGUAGE*, and *ORG_ID*.

> **Note:** The user information is defined by the `wsseUsername` property passed within the security headers. Detailed instructions on how to pass the security headers along with the SOAP request, see Passing Values to Security Headers, page 9-8.

The expected values for these elements are described in the following table:

Header Variables and Expected Values for Business Service Object Interfaces

Element Name	Expected Value
`RESPONSIBILITY_NAME`	`responsibility_name` (such as "System Administrator") or `{key} responsibility_key` (such as "{key} SYSTEM_ADMINISTRATOR")
`RESPONSIBILITY_APPL_NAME`	Application Short Name (such as "FND")
`SECURITY_GROUP_NAME`	Security Group Key (such as "STANDARD")
`NLS_LANGUAGE`	NLS Language (such as "AMERICAN")
`ORG_ID`	Org ID (such as "202")

> **Note:** NLS_Language and ORG_ID are optional values to be passed.

- If the NLS Language element is specified, SOAP requests can be consumed in the language passed. All corresponding SOAP responses and error messages can also be returned in the same language. If no language is identified, then the default language of the user will be used.
- If a service execution is dependent on any particular organization, then you must pass the ORG_ID element in the ServiceBean_Header of that SOAP request.

The context information can be specified by configuring an Assign activity before the Invoke activity in the BPEL PM.

Detailed information on how to set header variables for the SOAP request, see Assigning ServiceBean_Header Parameters, page 9-17.

Using Business Service Object WSDLs at Design Time

Based on the Purchase Order Service scenario described earlier, this section includes the following design-time tasks:

1. Create a SOA Composite Application with BPEL Process, page 9-5

   Use this step to create a new SOA Composite application with BPEL project called `ApprovePO.bpel`. This automatically creates two dummy activities - Receive and Reply - to receive input from a third party application and to reply output of the BPEL process to the request application.

2. Create a Partner Link, page 9-7

   Use this step to create a partner link for the PurchaseOrderService web service.

3. Add a Partner Link for File Adapter, page 9-9

   This is to synchronously read purchase order details received from the requestor.

4. Add Invoke Activities, page 9-13

   Use this step to create two Invoke activities in order to:

   1. Point to the File Adapter - Synchronous Read operation to read the purchase order from the input file.

   2. Point to the PurchaseOrderService web service to initiate the single purchase order approval process.

5. Add Assign Activities, page 9-16

   Use this step to create three Assign activities in order to:

   1. Set the SOAHeader details.

   2. Pass the purchase order details read from the File Adapter as an input to the Invoke activity for the PurchaseOrderService web service.

   3. Pass single purchase order approval information to the requestor through the Reply activity.

For general information and basic concept of a BPEL process, see Understanding BPEL Business Processes, page D-1 and *Oracle BPEL Process Manager Developer's Guide* for details.

Creating a New BPEL Project

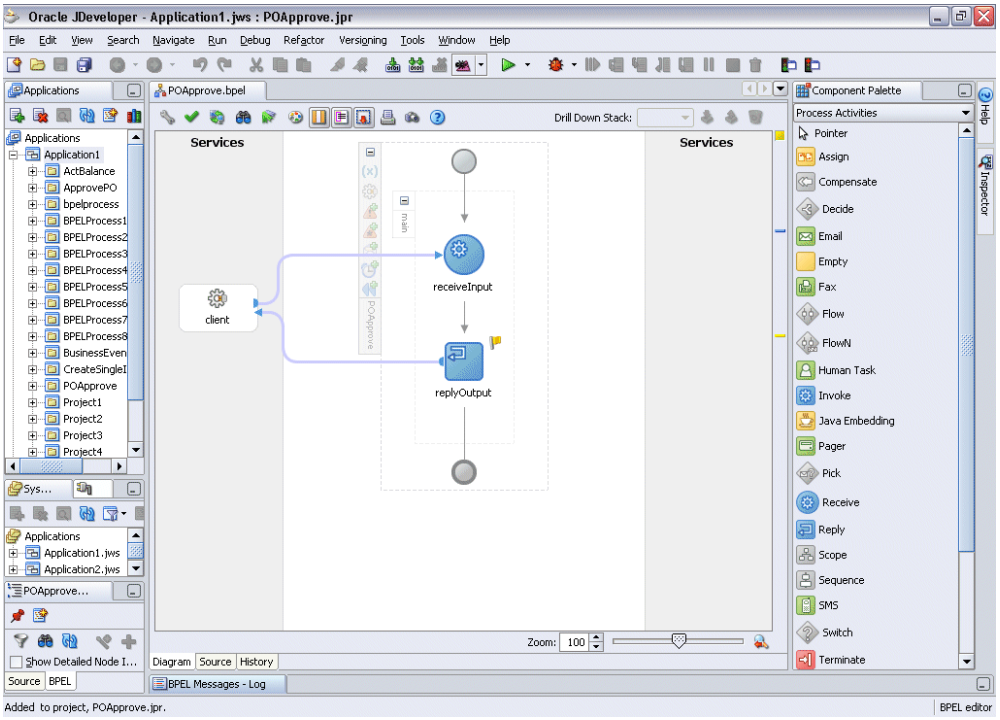
Use this step to create a new BPEL project that will contain various BPEL process activities.

To create a new BPEL project:

1. Open JDeveloper BPEL Designer.
2. From the **File** menu, select **New**. The New Gallery dialog box appears.
3. Select **All Items** from the **Filter By** box. This produces a list of available categories.
4. Expand the **General** node and then select **Projects**.
5. Select **BPEL Process Project** from the **Items** group.
6. Click **OK**. The BPEL Process Project dialog box appears.
7. In the **Name** field, enter a descriptive name for example `ApprovePO`.
8. From the Template list, select **Synchronous BPEL Process** and then select **Use Default Project Settings**.
9. Use the default input and output schema elements in the Input/Output Elements dialog box.
10. Click **Finish**.

    A new synchronous BPEL process is created with the Receive and Reply activities. The required source files including bpel.xml, using the name you specified (for example, `ApprovePO.bpel`) are also generated.

New BPEL Process Diagram



Creating a Partner Link

Use this step to configure a Partner Link called `PurchaseOrderService`.

To create a partner link for PurchaseOrderService:

1. In JDeveloper BPEL Designer, drag and drop the **PartnerLink** service from the Component Palette into the Partner Link border area of the process diagram. The Service Name dialog box appears.

2. Copy the WSDL URL corresponding to the PurchaseOrderService `/oracle/apps/fnd/framework/toolbox/tutorial/PurchaseOrderService` that you recorded earlier in the WSDL File field.

   A Partner Link Type message dialog box appears asking whether you want the system to create a new WSDL file that will by default create partner link types for you.

   Click **Yes** to have the `PurchaseOrderService` partner link created in the process diagram.

   The Partner Name and Partner Link Type values populated automatically. You can select the Partner Role value from the drop-down list.

   Click **Apply**.

3. **Passing Values to Security Headers**

   Select the Property tab and click the **Create Property** icon to select the following properties from the property name drop-down list in order to pass the security headers along with the SOAP request:

   - wsseUsername

     Specify the username to be passed in the Property Value box.

   - wssePassword

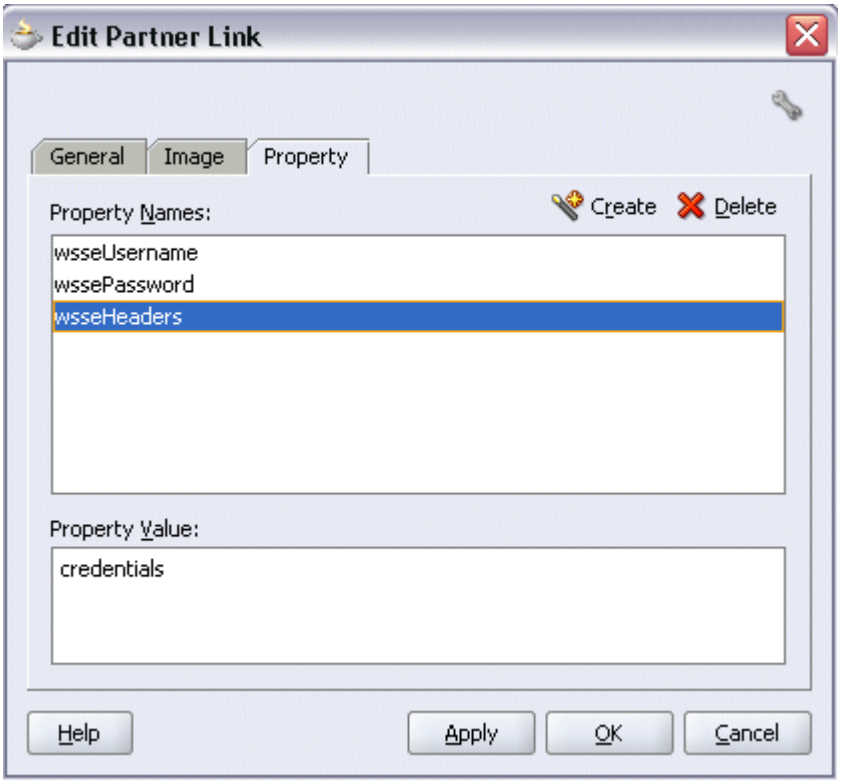
     Specify the corresponding password for the username to be passed in the Property Value box.

   - wsseHeaders

     Enter `credentials` as the property value.

   Click **Apply** to save the selected property values.

Edit Partner Link: Property Tab



4. Click **OK** to complete the partner link configuration. The partner link is created with the required WSDL settings, and is represented in the BPEL project by a new icon in the border area of the process diagram.

Adding a Partner Link for File Adapter

Use this step to synchronous read the purchase order details received from the third party application.

To add a Partner Link for File Adapter:

1. In JDeveloper BPEL Designer, drag and drop the **File Adapter** service from the **Adapter Service** section of the Component Palette into the Partner Link area of the process diagram. The Adapter Configuration Wizard welcome page appears.

2. Click **Next**. The Service Name dialog box appears.

3. Enter a name for the file adapter service for example `ReadPO`. You can add an optional description of the service.

4. Click **Next** and the Operation dialog box appears.

5. Specify the operation type, for example **Synchronous Read File**. This automatically populates the **Operation Name** field.

   Click **Next** to access the File Directories dialog box.

6. Select **Physical Path** radio button and enter the input payload file directory information. For example, enter `/usr/tmp/` as the directory name.

   Uncheck the **Delete Files after successful retrieval** check box. Click **Next** to open the File Name dialog box.

7. Enter the name of the file for the synchronous read file operation. For example, enter 'Input.xml'. Click **Next**. The Messages dialog box appears.

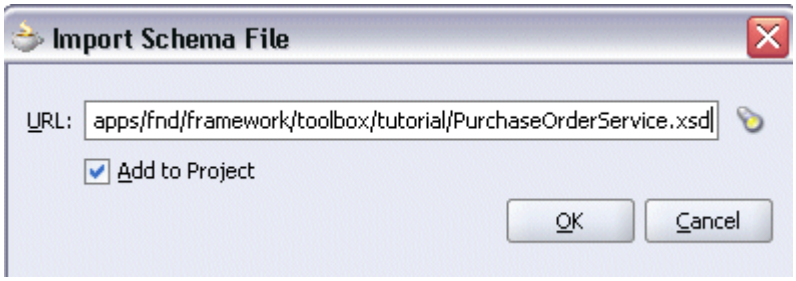
8. Select **Browse** for schema file in Schema Location. The Type Chooser window is displayed.

   1. Click **Import Schema Files** button on the top right corner of the Type Chooser window.

   2. Enter the schema location for the service. Such as `http://<myhost>:<port>/webservices/AppsWSProvider/oracle/apps/fnd/framework/toolbox/tutorial/PurchaseOrderService.xsd`.

      Schema location for your service can be found from the service WSDL URL (for example, `http://<myhost>:<port>/AppsWSProvider/oracle/apps/fnd/framework/toolbox/tutorial/PurchaseOrderService?wsdl`).

Import Schema File Dialog



3. Select the **Add to Project** check box and click **OK**.

4. Click **OK** for Import schema prompt.

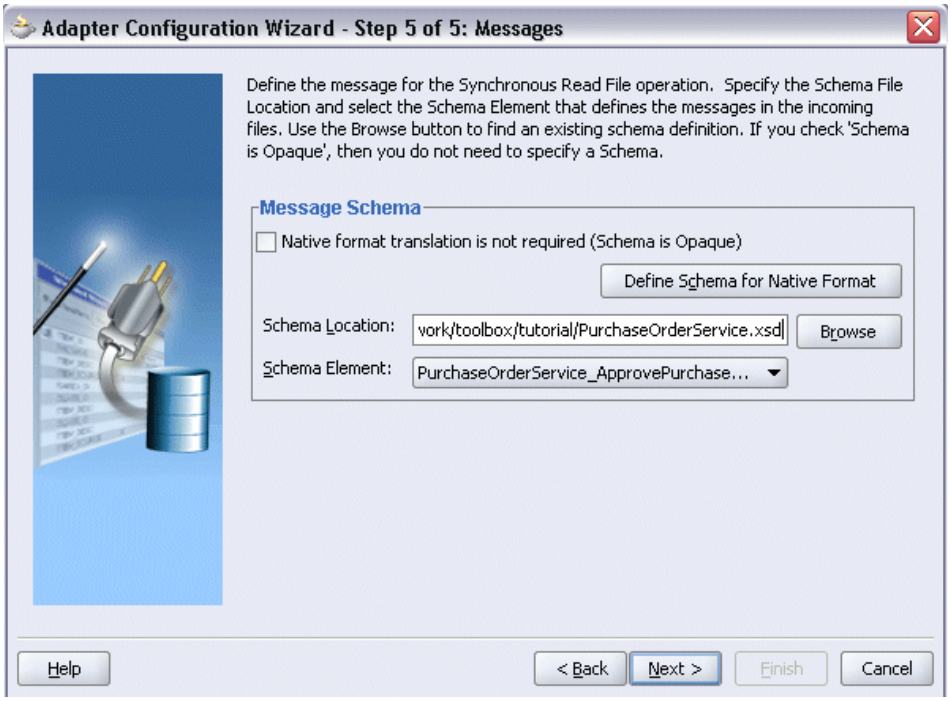
   The Imported Schemas folder is automatically added to the Type Chooser window.

5. Select schema element by expanding the Imported Schemas folder > PurchaseOrderService.xsd > PurchaseOrderService_ApprovePurchaseOrder.

   Click **OK**.

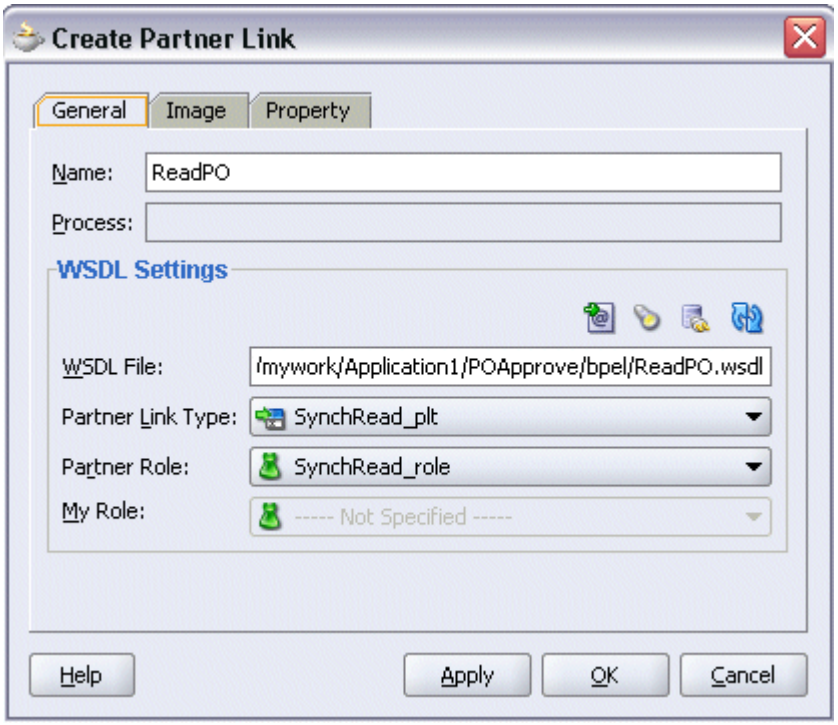
   The selected schema location and element values are displayed.

Message Dialog with Selected Message Schema



9. Click **Next** and then **Finish**. The wizard generates the WSDL file corresponding to the partner link. The main Create Partner Link dialog box appears, specifying the new WSDL file `ReadPO.wsdl`.

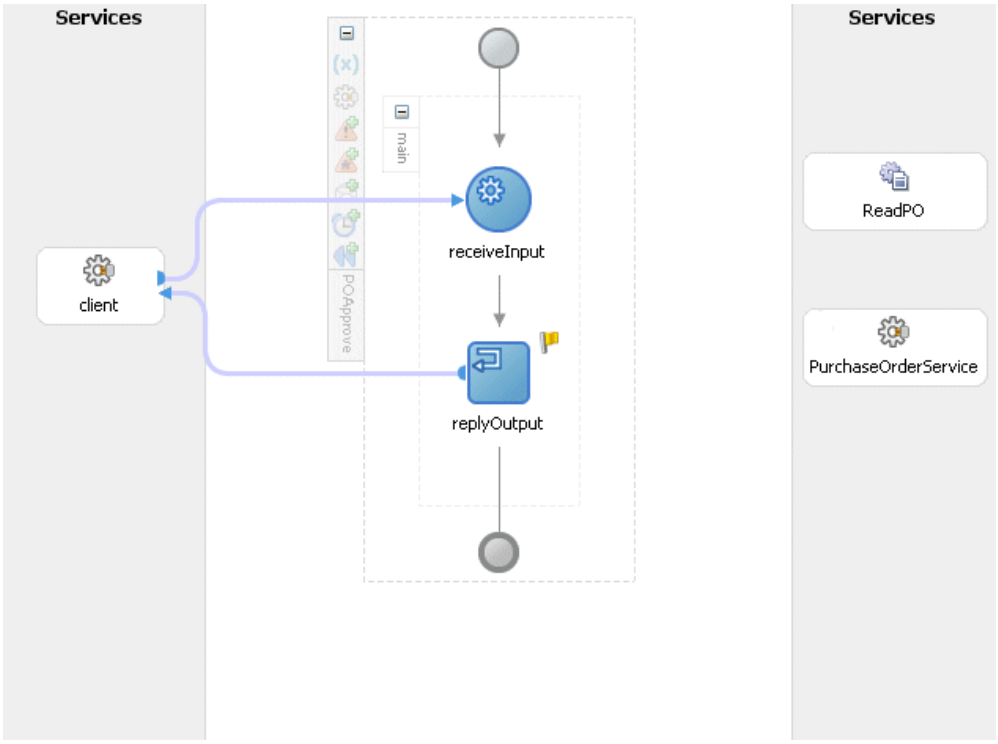
Create Partner Link Dialog



Click **Apply** and then **OK** to complete the configuration and create the partner link with the required WSDL settings for the File Adapter Service.

The `ReadPO` Partner Link appears in the BPEL process diagram.

BPEL Process Dialog with Partner Link for File Adapter



Adding an Invoke activity

This step is to configure two Invoke activities:

1. Point to the File Adapter `ReadPO` to synchronous read the purchase order from the Receive activity.

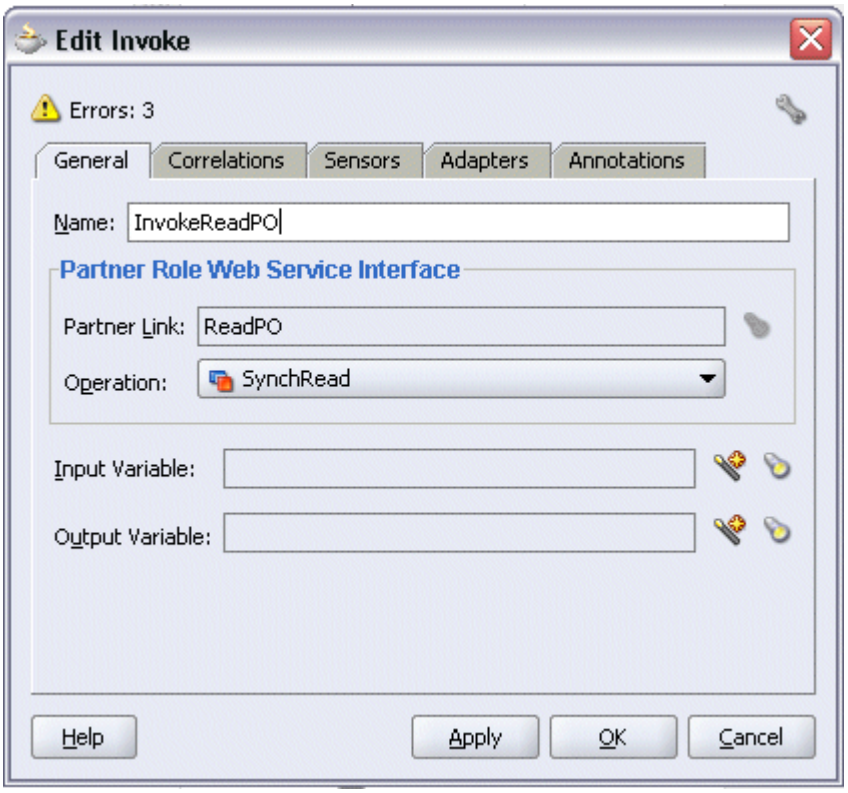
2. Point to the `PurchaseOrderService` partner link to send the transaction information that is received from the Assign activities to initiate the single purchase order approval process.

To add an Invoke activity for ReadPO Partner Link:

1. In JDeveloper BPEL Designer, drag and drop the **Invoke** activity from the Component Palette into the Activity box of the process diagram, between the **Receive** and **Reply** activities.

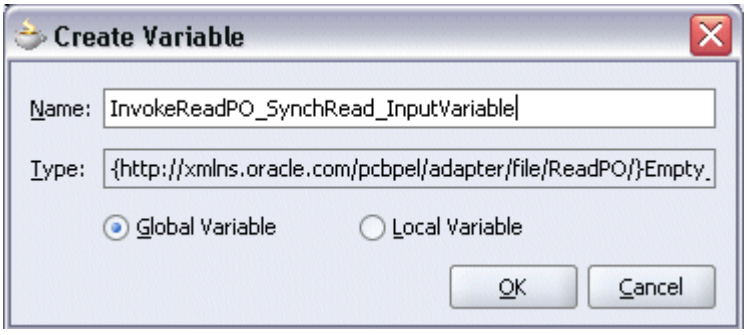
2. Link the Invoke activity to the `ReadPO` service. The Invoke activity will send event data to the partner link. The Edit Invoke dialog box appears.

Edit Invoke Dialog



3. Enter a name for the Invoke activity, and then click the **Create** icon next to the **Input Variable** field to create a new variable. The Create Variable dialog box appears.

Create Variable Dialog



4. Select **Global Variable** and then enter a name for the variable. You can also accept the default name. Click **OK**.

5. Enter a name for the Invoke activity and then click the **Create** icon next to the

Output Variable field to create a new variable. The Create Variable dialog box appears.

6. Select **Global Variable** and then enter a name for the variable. You can also accept the default name. Click **OK**.

7. Click **Apply** and then **OK** in the Edit Invoke dialog box to finish configuring the Invoke activity.

   The Invoke activity appears in the process diagram.

To add an Invoke activity for PurchaseOrderService Partner Link:

1. In JDeveloper BPEL Designer, drag and drop the **Invoke** activity from the Component Palette into the Activity box of the process diagram, after the **Invoke** and **Reply** activities.

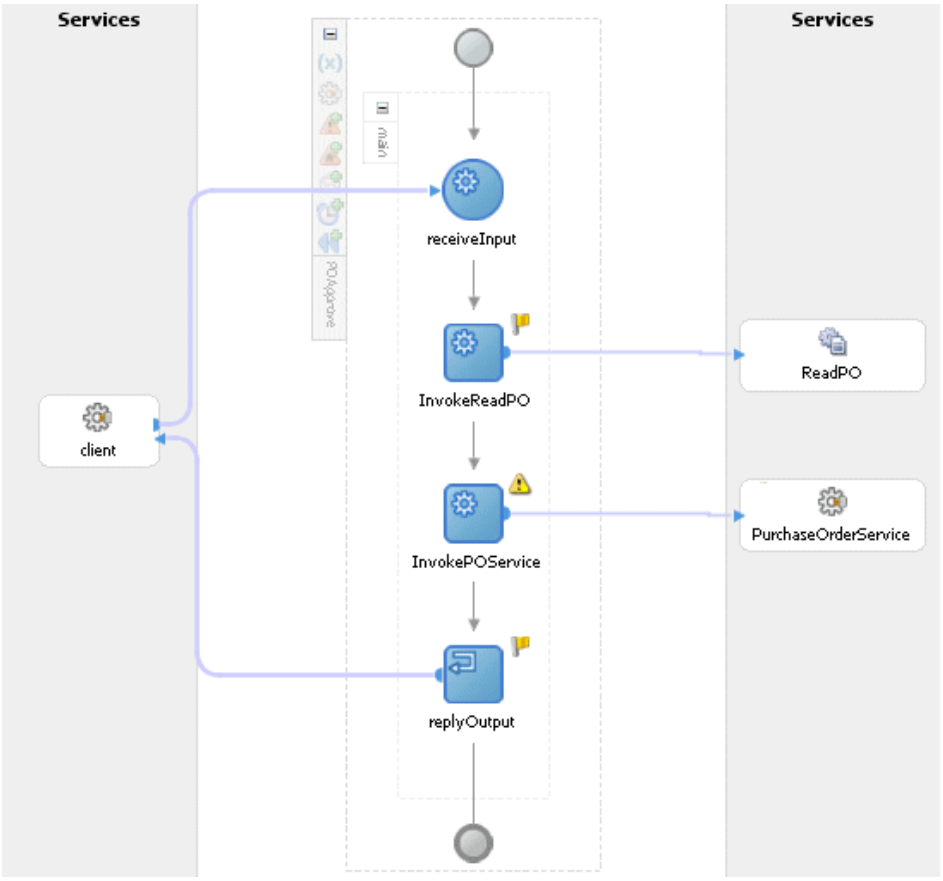
2. Link the Invoke activity to the `PurchaseOrderService` service. The Invoke activity will send event data to the partner link. The Edit Invoke dialog box appears.

3. Enter a name for the Invoke activity such as 'InvokePOService'. Create input and output variables described in the first Invoke activity. Click **OK** to close the Create Variable dialog box.

4. Click **Apply** and then **OK** to finish configuring the Invoke activity.

   The Invoke activity appears in the process diagram.

BPEL Process Dialog with Invoke Activities



Adding an Assign activity

This step is to configure three Assign activities:

1. To set the header details.

   Note: You need to populate certain variables in the BPEL process for ServiceBean_Header elements to pass values that may be required to embed application context into SOAP envelopes for Web service authorization. These ServiceBean_Header elements for Business Service Object interface type are *RESPONSIBILITY_NAME*, *RESPONSIBILITY_APPL_NAME*, *SECURITY_GROUP_NAME*, *NLS_LANGUAGE*, and *ORG_ID*.

   Detailed information on how to set ServiceBean_Header for the SOAP request, see Assigning ServiceBean_Header Parameters, page 9-17.

2. To pass the purchase order details read from the File Adapter as an input to the second Invoke activity for `PurchaseOrderService` partner link.

3. To pass single purchase order approval information to the requestor through the dummy Reply activity.

To add the first Assign activity to pass header details:

Assigning ServiceBean_Header Parameters:

1. In JDeveloper BPEL Designer, drag and drop the **Assign** activity from the Component Palette into the Activity box of the process diagram, between the two **Invoke** activities.

2. Double-click the **Assign** activity to access the Edit Assign dialog box.

3. Click the General tab to enter the name for the Assign activity, such as 'SetServiceBeanHeader'.

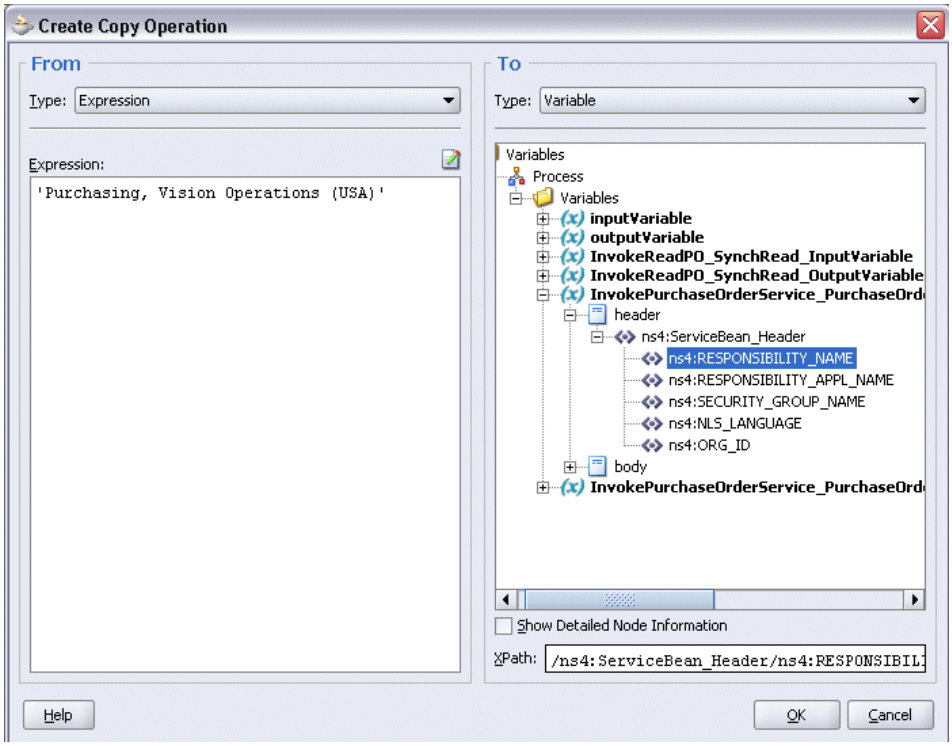
4. On the Copy Operation tab, click **Create** and then select **Copy Operation** from the menu. The Create Copy Operation window appears.

5. Enter the first pair of parameters:

   - In the From navigation tree, select type Expression and then enter 'Purchasing, Vision Operations (USA)' in the Expression box.

   - In the To navigation tree, select type Variable. Navigate to **Variable > Process > Variables > InvokePurchaseOrderService_PurchaseOrderService_AcknowledgePurchaseOrder_InputVariable > header> ns4:ServiceBean_Header** and select **ns4: RESPONSIBILITY_NAME**. The XPath field should contain your selected entry.

Create Copy Operation Dialog for Assigning Responsibility Parameter Value



- Click **OK**.

6. Enter the second pair of parameters by selecting **Copy Operation** from the Create drop-down list with the following values:

   - In the From navigation tree, select type Expression and then enter 'PUR' in the Expression box.

   - In the To navigation tree, select type Variable. Navigate to **Variable > Process > Variables > InvokePurchaseOrderService_PurchaseOrderService_AcknowledgePurchaseOrder_InputVariable > header> ns4:ServiceBean_Header** and select **ns4:RESPONSIBILITY_APPL_NAME**. The XPath field should contain your selected entry.

   - Click **OK**.

7. Enter the third pair of parameters:

   - In the From navigation tree, select type Expression and then enter 'STANDARD' in the Expression box.

- In the To navigation tree, select type Variable. Navigate to **Variable > Process > Variables > InvokePurchaseOrderService_PurchaseOrderService_AcknowledgePurchaseOrder_InputVariable > header> ns4:ServiceBean_Header** and select **ns4: SECURITY_GROUP_NAME**. The XPath field should contain your selected entry.

- Click **OK**.

8. Enter the fourth pair of parameters:

   - In the From navigation tree, select type Expression and then enter 'AMERICAN' in the Expression box.

   - In the To navigation tree, select type Variable. Navigate to **Variable > Process > Variables > InvokePurchaseOrderService_PurchaseOrderService_AcknowledgePurchaseOrder_InputVariable > header> ns4:ServiceBean_Header** and select **ns4: NLS_LANGUAGE**. The XPath field should contain your selected entry.

   - Click **OK**.

9. Enter the fifth pair of parameters by selecting **Copy Operation** from the Create drop-down list with the following values:

   - In the From navigation tree, select type Expression and then enter '204' in the Expression box.

   - In the To navigation tree, select type Variable. Navigate to **Variable > Process > Variables > InvokePurchaseOrderService_PurchaseOrderService_AcknowledgePurchaseOrder_InputVariable > header> ns4:ServiceBean_Header** and select **ns4: ORG_ID**. The XPath field should contain your selected entry.

   - Click **OK**.

10. The Edit Assign dialog box appears.

11. Click **Apply** and then **OK** to complete the configuration of the Assign activity.

To enter the second Assign activity to pass PO details to the InvokePOService Invoke activity:

1. In JDeveloper BPEL Designer, drag and drop the **Assign** activity from the Component Palette into the Activity box of the process diagram, between the **Assign** and **Invoke** activities.

2. Double-click the **Assign** activity to access the Edit Assign dialog box.

3. Click the General tab to enter the name for the Assign activity, such as 'SetPOApproval'.

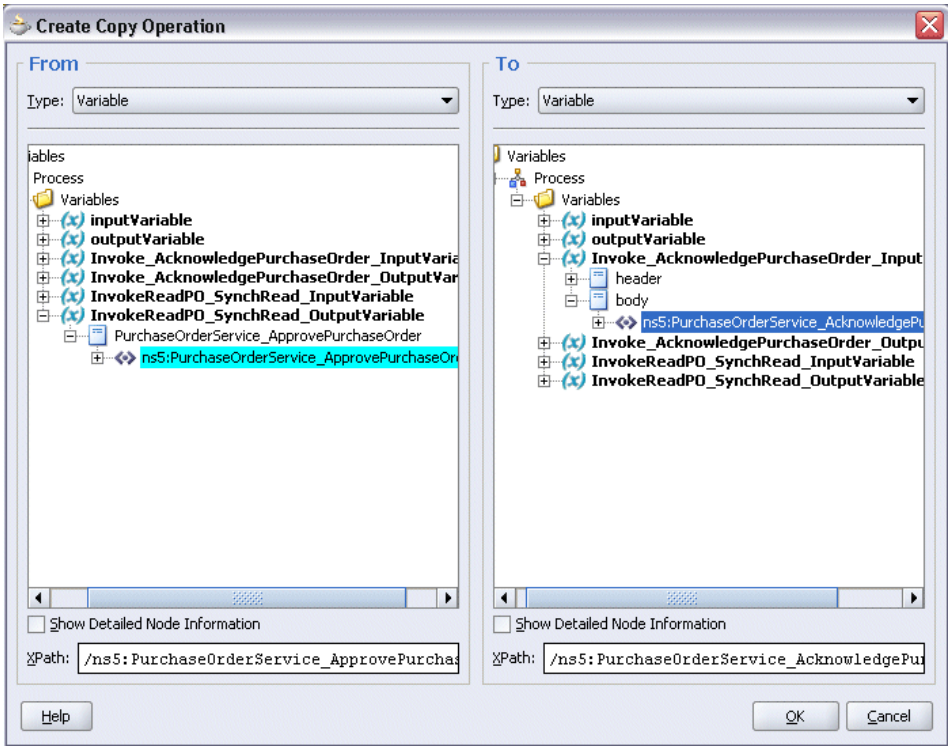
4. On the Copy Operation tab, click **Create** and then select **Copy Operation** from the menu. The Create Copy Operation window appears.

5. Enter the following information:

   - In the From navigation tree, navigate to **Variable > Process > Variables > InvokeReadPO_SynchRead_OutputVariable > PurchaseOrderService_ApprovePurchaseOrder >** and select **ns5: PurchaseOrderService_ApprovePurchaseOrder**. The XPath field should contain your selected entry.

   - In the To navigation tree, select type Variable. Navigate to **Variable > Process > Variables > InvokePurchaseOrderService_PurchaseOrderService_AcknowledgePurchaseOrder_InputVariable > body** and select **ns5: PurchaseOrderService_AcknowledgePurchaseOrder**. The XPath field should contain your selected entry.

Create Copy Operation Dialog



- Click **OK**.

6. Click **Apply** and then **OK** to complete the configuration of the Assign activity.

To enter the third Assign activity to set the SOAP response to output:

1. Add the second Assign activity by dragging and dropping the **Assign** activity from the Component Palette into the Activity box of the process diagram, between the `InvokePOService` **Invoke** and the **Reply** activities.

2. Repeat Step 2 to Step 4 described in creating the first Assign activity to add the second Assign activity called 'SetPOStatus'.

3. Enter the following information:

   - In the From navigation tree, select type Variable. Navigate to **Variable > Process > Variables > InvokePurchaseOrderService_PurchaseOrderService_AcknowledgePurchase Order_OutputVariable** and select **body**.

   - In the To navigation tree, select type Variable. Navigate to **Variable > Process > Variables > outputVariable** and select **payload**.

- Click **OK**.

  The Edit Assign dialog box appears.

4. Click **Apply** and then **OK** to complete the configuration of the Assign activity.

Deploying and Testing the BPEL Process at Runtime

After creating a BPEL process using the WSDL URL generated from the business service object interface definition, you can deploy it to a BPEL server if needed. To ensure that this process is modified or orchestrated appropriately, you can also manually test the BPEL process by initiating the business process contained in the BPEL process to test the interface integration.

Prerequisites

Before deploying the BPEL process using Oracle JDeveloper, you must ensure that you have established the connectivity between the deign-time environment and the runtime servers including the application server and the integration server.

How to configure the necessary server connection, see Configuring Server Connection, page B-1.

To validate your BPEL process, perform the following runtime tasks:

1. Deploy the BPEL process, page 9-22

   Once you deploy the process to a BPEL server, it becomes available so that you can run the process manually to test it for validation.

2. Test the BPEL process, page 9-23

   After deploying a BPEL process, you can manage the process from the BPEL console to manually initiate the business process and test the interface integration contained in your BPEL process.

Deploying the BPEL Process

You must deploy the Approve Purchase Order BPEL process (`POApprove.bpel`) that you created earlier before you can run it.

To deploy the BPEL process:

1. In the Applications Navigator of JDeveloper BPEL Designer, select the **POApprove** project.

2. Right-click the project and click **Make** action from the menu.

   Look for any compilation error messages in Message Log.

   Right-click the project and select **Deploy >*Integration Server Connection name*> Deploy to Default Domain** action from the menu.

For example, you can select **Deploy > BPELServerConn > Deploy to Default Domain** to deploy the process if you have the BPEL Process Manager setup appropriately.

3. Look for 'Build successful' message in Apache Ant – Log to ensure that the BPEL project is compiled and successfully deployed.

Testing the BPEL Process

Once the BPEL process is deployed, it can be seen in the BPEL console. You can manage and monitor the process from the BPEL console. You can also test the process and the integration interface by logging on to Oracle E-Business Suite to manually initiate the processes.

To test the BPEL process:

1. Log in to Oracle Application Server 10g BPEL Console (`http://<soaSuiteServerHostName>:<port>/BPELConsole`). The BPEL Console login page appears.

2. Enter the username and password and click **Login**.

   The Oracle Enterprise Manager 10g BPEL Control appears.

3. In the BPEL Console, confirm that 'POApprove' has been deployed.

4. Click the ApprovePO link to open the Initiate tab

5. Enter the payload input field and click **Post XML Message** to initiate the process.

6. The audit trail provides information about the steps that have been executed. You can check the audit trail by clicking the **Audit Instance** icon.

   This is to verify that a purchase order is approved successfully.

Validating the Process in Oracle E-Business Suite

Additionally, you can validate the BPEL process in Oracle E-Business Suite. Log in to Oracle E-Business Suite with the Purchasing responsibility. Open up the Purchase Orders form and search for the supplier to bring up the purchase order details. Notice that the Status field is 'Approved'.

Using Business Service Object REST Services

REST Service Invocation Scenario

To explain how to use a Business Service Object REST service, this scenario takes an interface "Location Business Object Services" (`oracle.apps.ar.hz.service.party.LocationService`) as an example to guide you through the Business Service Object REST service invocation.

At runtime when a request of providing location business object details is received, the `getLocation` service operation contained in the "Location Business Object Services" interface is invoked to fetch the location details of that particular location business object from TCA.

After a successful service invocation, the REST response message that contains the specific location business object details is sent back to the requestor.

Invoking a Business Service Object REST Service

Based on the REST service invocation scenario described here, this chapter includes the following topics:

1. Deploying a Business Service Object REST Service, page 9-24
2. Creating a Security Grant for the Deployed Service, page 9-26
3. Recording Resource Information from Deployed WADL, page 9-27
4. Invoking a REST Service Using Command Lines, page 9-27

Deploying a Business Service Object REST Service

Use the following steps to deploy the Business Service Object "Location Business Object Services" (`oracle.apps.ar.hz.service.party.LocationService`) as a REST service:

1. Log in to Oracle E-Business Suite as a user who has the Integration Administrator role.

   Select the Integrated SOA Gateway responsibility and the Integration Repository link from the navigation menu.

2. In the Integration Repository tab, click **Search** to access the main Search page.

3. Enter the following key search values as the search criteria:

   - Internal Name: Location Business Object Services
   - Interface Type: Business Service Object

4. Click **Go** to run the search.

   Click the "Location Business Object Services" interface name link to open the interface details page.

5. In the REST Web Service tab, enter the following information:

   - Service Alias: location

     The alias will be displayed as the service endpoint in the WADL and schema for

the selected method or operation.

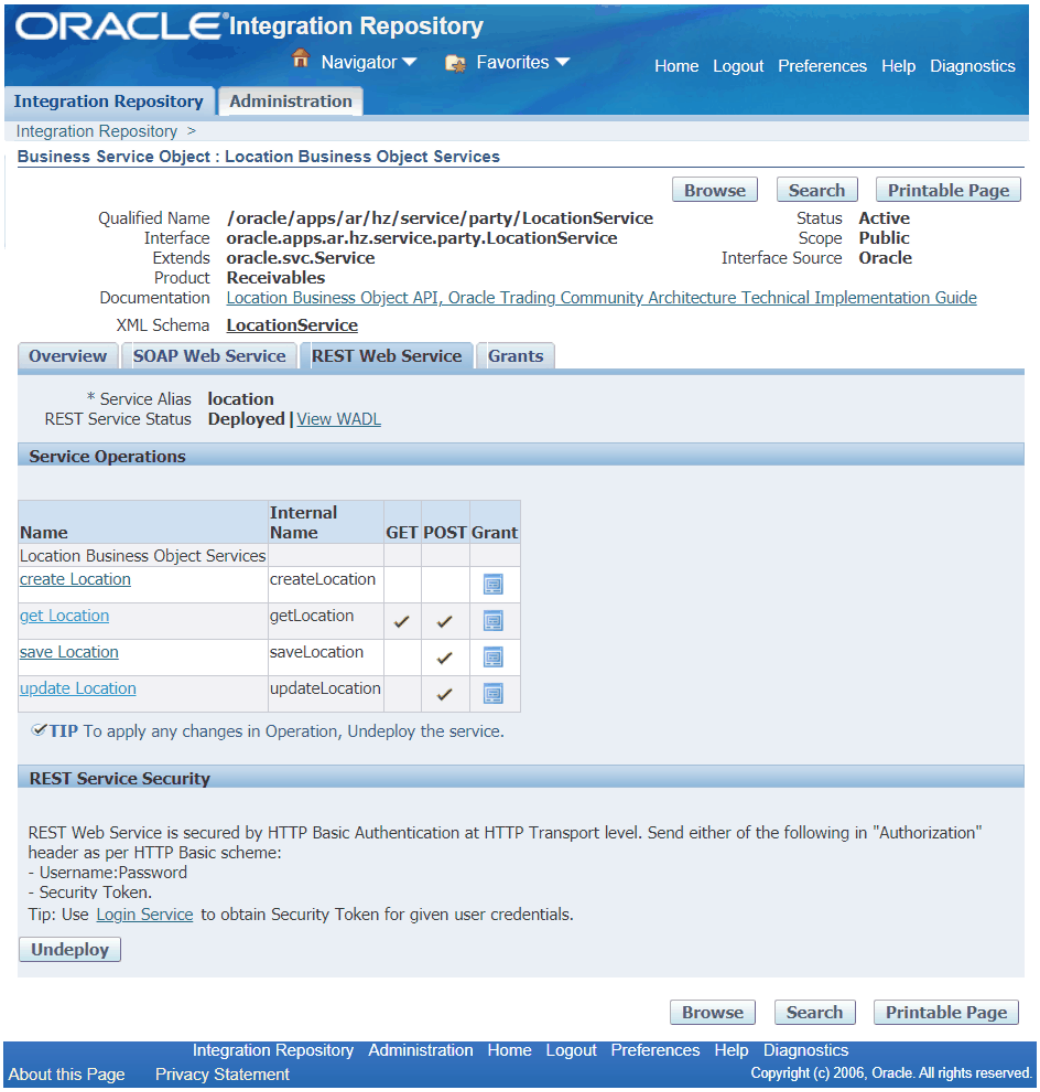
- Select Desired Service Operations

  In the second row `getLocation`, select both the GET and POST HTTP method check boxes.

6. Click **Deploy** to deploy the service to an Oracle E-Business Suite WebLogic environment.

Once the REST service has been successfully deployed, 'Deployed' appears in the REST Service Status field along with the **View WADL** link. Click the **View WADL** link to view the deployed service WADL description.

Interface Details Page with REST Web Service Tab



For more information on deploying REST services, see Deploying REST Web Services, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

Creating a Security Grant for the Deployed Service

After deploying the "Location Business Object Services" as a REST service, the integration administrator can create a security grant to authorize the service or method access privileges to a specific user, a user group, or all users.

Use the following steps to create a security grant:

1. Log in to Oracle E-Business Suite as a user who has the Integration Administrator role. Select the Integrated SOA Gateway responsibility and the Integration

Repository link from the navigation menu.

2. Perform a search to locate the "Location Business Object Services" service the administrator just deployed earlier.

3. In the interface details page of the selected "Location Business Object Services", click the Grants tab.

4. Select the `getLocation` method check box and then click **Create Grant**.

5. In the Create Grants page, select "All User" as the Grantee Type.

   Note: In this example, the `getLocation` service operation is granted to all users. In actual implementation, you should define strict security rules. Create grant to a user or more appropriately to a group of users defined by roles and responsibilities.

   Click **Create Grant**. This grants the selected method access privilege to all Oracle E-Business Suite users.

Recording Resource Information from Deployed WADL

To obtain service resource information from the deployed "Location Business Object Services" service, an integration developer clicks the **View WADL** link in the REST Web Service tab. This displays the WADL description in a different window.

Copy or record the REST service endpoint from the WADL description. This will be used later when invoking the service.

`http://<hostname>:<port>/webservices/rest/location/getLocation/`

Replace `location`, the alias in this sample deployment, with the alias name you used in your deployment.

Invoking a REST Service Using Command Lines

In this example, the deployed `getLocation` service operation with the GET method will be invoked to fetch data.

A third party command line tool called `cURL` is used to transfer data using URL syntax. This tool is available by default on most Linux environments.

Note: You can test the REST service invocation using any REST client.

10

Using Composite Services - BPEL

Overview

Composite services use native services as building blocks to orchestrate the business invocation sequence from discreate web services into a meaningful end-to-end business flow through a web service composition language BPEL. Strictly speaking, this type of interface is comparatively service enabled without additional service generation process as required by native interface types.

At design time, based on business needs, an integration developer can create a composite service - BPEL type by using any of the web service WSDL URL that has been successfully generated and deployed to Oracle Application Server.

At run time, the developer can also view each composite service details by selecting an appropriate composite service from the Oracle Integration Repository browser, download the selected composite service from the repository to their local directories, open them in Oracle JDeveloper to modify the BPEL project if necessary before deploying it to a BPEL server in Oracle SOA Suite or a third party BPEL PM server.

This chapter discusses each runtime task listed below for using BPEL composite services. Detailed design-time tasks on how to create a BPEL composite service are included in each individual interface described earlier in this book.

For general information and basic concept of a BPEL process, see Understanding BPEL Business Processes, page D-1.

Viewing Composite Services

Similar to all other users, system integration developers can view a composite service by navigating to the Composite Service interface type directly from the Oracle Integration Repository Browser window or by performing a search by selecting Composite Service interface type in the Search page.

Clicking on a composite service name link from the navigation tree or search results, you will find the composite service interface details page where displays composite service name, description, BPEL file, and other annotated information.

The composite service details page allows you to perform the following tasks in the BPEL Files region:

- View an abstract WSDL file by clicking the URL link
- Review XML representation file by clicking the URL link

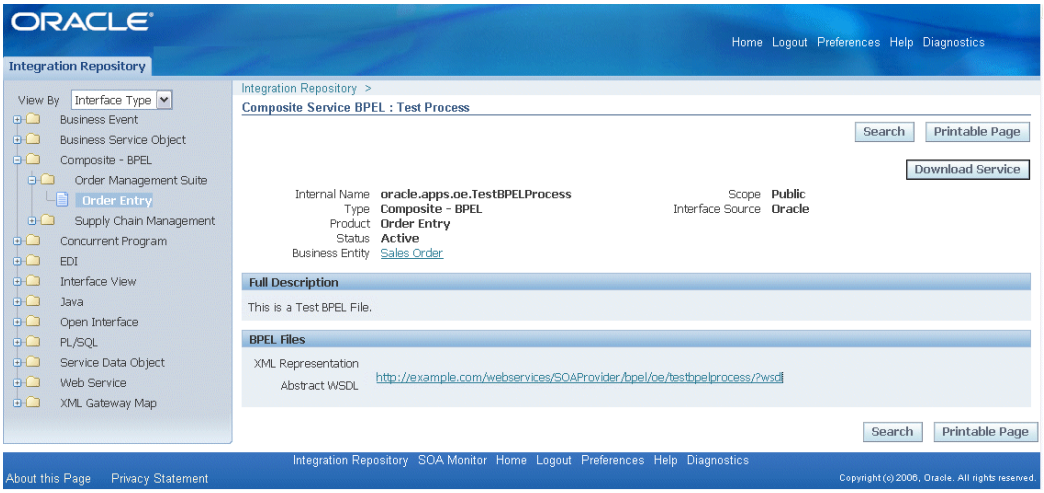
You can also download a corresponding composite service project file, such as BPEL file, to your local machine. See: Downloading Composite Services, page 10-2.

Downloading Composite Services

In addition to viewing composite service details and reviewing a WSDL abstract, the developers can download the composite service relevant files aggregated in a .JAR file to your local machine.

> **Important:** In general, only system integration developers and integration repository administrators can download the composite services. However, general users (system integration analysts) who are granted the download privilege, an Integration Repository Download Composite Service permission set FND_REP_DOWNLOAD_PERM_SET, can also perform the download action.
>
> For more information on how to grant Download Composite Service privilege, see Role-Based Access Control (RBAC) Security, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

Composite Service Interface Details Page



To download the .ZIP file for a composite service, navigate to the composite service details page for a service that you want to download, and then click **Download Service** to download the file to your local machine.

After you download the file, you can unzip the BPEL .JAR file and open the BPEL process in Oracle JDeveloper for further modification on service endpoints if needed. Additionally, You can deploy the BPEL process to a BPEL server through Oracle BPEL Process Manager. For information on modifying and deploying BPEL projects, see Modifying and Deploying BPEL Processes, page 10-4.

To download a composite service:

1. Log in to Oracle Integration Repository as a user who has the System Integration Developer role.

   Select the Integrated SOA Gateway responsibility from the navigation menu. Select the Integration Repository link.

2. In the Integration Repository tab, select 'Interface Type' from the View By drop-down list.

3. Expand the Composite Service interface type node to locate your desired composite service.

4. Click the composite service that you want to download it to open the Composite Service Interface Details page.

5. Click **Download Service** to download the selected composite file to your local machine.

Modifying and Deploying BPEL Processes

After downloading a composite service BPEL project, an integration developer can optionally modify the BPEL project. This can be done by first unzipping the BPEL .JAR file and then opening the BPEL file in Oracle JDeveloper to modify the BPEL process endpoints if necessary.

Additionally, the BPEL process can be further deployed to a BPEL server in Oracle SOA Suite BPEL PM or a third party BPEL PM in a J2EE environment. To ensure that this process is modified or orchestrated appropriately, you can manually test the BPEL process by initiating the business process contained in the BPEL process to test the interface integration.

The modification of a BPEL process uses the similar logic during the BPEL process creation. See Understanding BPEL Business Processes, page D-1 and design-time tasks for each interface type discussed earlier in this book.

How to test and validate the BPEL process that contains an interface exposes as a Web service, refer to the run-time tasks of the interface type described in this book.

For BPEL process modification and deployment described in this section, we use Oracle JDeveloper 10.1.3.3.0 to modify the BPEL process and use Oracle SOA Suite BPEL server 10.1.3.3.0 for the process deployment.

To modify a BPEL process:

1. Open a BPEL file in Oracle JDeveloper BPEL Designer.
2. From the **File** menu, select **Open**.
3. Locate your BPEL file from the directory that you want to modify. Click **Open** in the Open window.
4. The selected BPEL process diagram appears.
5. Modify the BPEL process endpoints if necessary.
6. Save your work.

To deploy a BPEL process:

1. In the Applications Navigator of JDeveloper BPEL Designer, select the BPEL project that you want to deploy.
2. Right-click the project and select **Deploy** action from the menu. Click on **Invoke Deployment Tool** and enter your BPEL Process Manager information.

   For example, you can select **Deploy > BPELServerConn > Deploy to Default Domain** to deploy the process if you have the BPEL Process Manager setup appropriately.

3. The Password Prompt dialog box appears.

   Enter the password for the default domain in the **Domain Password** field and click **OK**.

   The BPEL project is compiled and successfully deployed.

11

Creating and Using Custom Integration Interfaces

Overview

To support custom integration interfaces, Oracle E-Business Suite Integrated SOA Gateway provides a mechanism allowing these custom interfaces to be displayed through the Oracle Integration Repository browser along with Oracle packaged interfaces. This enables Oracle Integration Repository a single source of truth in centrally displaying all integration interfaces regardless of custom or Oracle packaged ones within the entire Oracle E-Business Suite.

Custom interface definitions can be created for various interface types including custom interface definitions for XML Gateway Map, Business Event, PL/SQL, Concurrent Program, Business Service Object, Java APIs, Java Bean Services, Application Module Services, and Composite Service for BPEL type.

> **Note:** Please note that custom interface types of EDI, Open Interface Tables, and Open Interface Views are not supported in this release.
>
> Oracle Integration Repository currently does not support the creation of custom Product Family and custom Business Entity.

Depending on your business needs, integration developers can create custom integration interfaces first, and then annotate the custom interfaces based on the Integration Repository annotation standards. Once these custom interfaces are annotated, appropriate validation on the annotated custom interfaces is required before they are uploaded to the Integration Repository by an integration repository administrator.

For custom integration interfaces of interface types

If a custom interface created for a supported interface type has been uploaded to Oracle Integration Repository, to use this custom interface, an integration repository

administrator should first create necessary security grants, and then generate and deploy the Web service WSDL file to the application server if the custom interface type can be service enabled. Thus, the deployed service can be exposed to customers through a service provider and invoked through any of the Web service clients or orchestration tools.

For custom composite service - BPEL type

If a custom interface is needed for a composite service - BPEL type, the integration developer will first create a composite service by orchestrating discrete native services into a meaningful process flow using BPEL. Based on the annotation standards specifically for composite service, the developer will then annotate the composite service, and create and unzip the JAR file of the BPEL project. Like custom interfaces of other interface types, appropriate validation on the BPEL project JAR file is required to ensure its compliance with the annotation standards before it is uploaded to the Integration Repository.

To have a better understanding of how to create custom interfaces as well as how to use custom interfaces as Web services, the following topics are discussed in this chapter:

Creating Custom Integration Interfaces

The following topics are discussed in this section:

Creating Custom Integration Interfaces of Interface Types

Custom interface definitions can be created and annotated for almost all interface types. After appropriate validation, these custom interfaces will be uploaded to Oracle Integration Repository and embedded into the interface categories where they belong.

> **Note:** Custom interface types of EDI, Open Interface Tables, and Open Interface Views are not supported in this release.
>
> Oracle Integration Repository currently does not support the creation of custom Product Family and custom Business Entity.

Enabling Custom Integration Interfaces

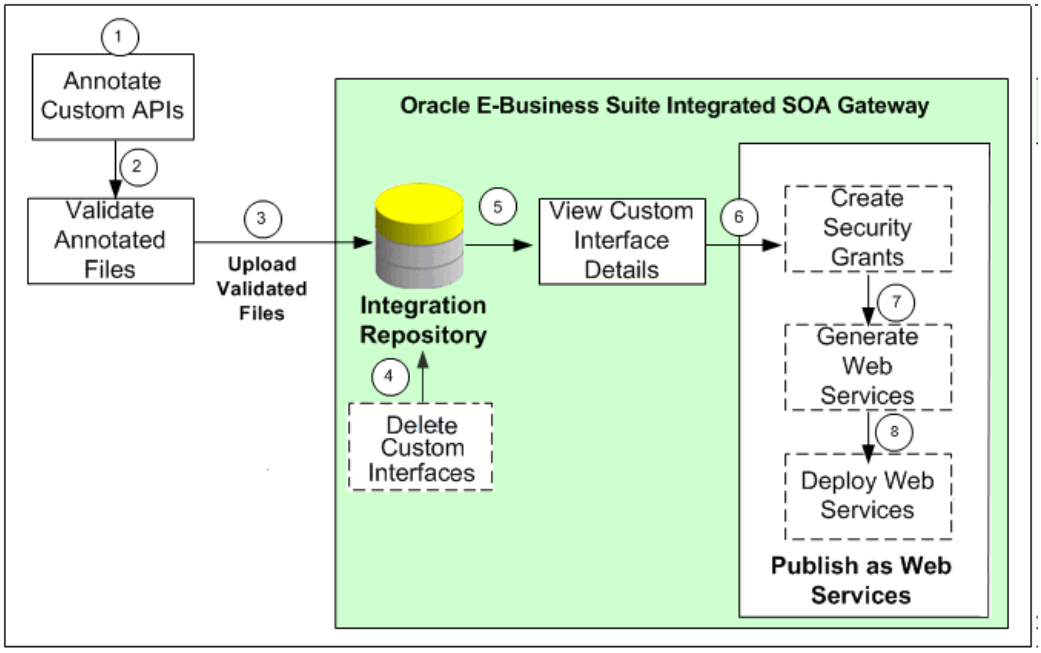
The custom interface design and service enablement process flow can be illustrated in the following diagram:

Note: Not all integration interface definitions can be service enabled. Oracle Integration Repository supports service enablement only for the following interface types:

- PL/SQL
- XML Gateway Map (inbound)
- Concurrent Program
- Business Service Object (Service Beans)
- Java Bean Services
- Application Module Services

Note that the Business Event and XML Gateway Map (outbound) interface types are supported through subscription model.

Custom Integration Interfaces Development Process Flow



1. Users who have the System Integration Developer role can annotate custom integration interface definition based on the Integration Repository annotation standards for the supported interface types.

   See: Creating and Annotating Custom Integration Interfaces, page 11-6.

2. Users who have the Integration Repository Administrator role can validate the annotated custom interface definitions against the annotation standards. This validation is performed by executing the Integration Repository Parser (IREP Parser), a design-time tool, to read the annotated files and then generate an Integration Repository loader file (iLDT) if no error occurred.

   For information on how to generate and upload the iLDT files, see Generating and Uploading iLDT Files, page 11-9.

3. Users who have the Integration Repository Administrator role can upload the generated iLDT file to Oracle Integration Repository.

   See: Uploading ILDT Files to Integration Repository, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

4. (Optional) Users who have the Integration Repository Administrator role can delete the custom interfaces if needed.

   Before starting to use a custom integration interface from the Integration Repository, users who have the Integration Repository Administrator role can delete the custom interface if it is not yet generated or deployed as a web service.

The administrators can first locate the custom interface from the Integration Repository user interface, and then click **Delete Interface** in the Overview tab of the custom interface details page.

If a custom interface has been deployed, it must be undeployed first before it can be deleted. That is, its web service status must be either 'Generated' for a custom SOAP service or 'Not Deployed' for a custom REST service. See: Deleting Custom Integration Interfaces, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

5. All users can view the uploaded custom interfaces from the Integration Repository user interface.

6. (Optional) Users who have the Integration Repository Administrator role then create necessary security grants for the custom integration interfaces if needed.

   This is achieved by first locating the custom interface from the Integration Repository, and then selecting methods contained in the selected custom interface before clicking **Create Grant**. The Create Grants page is displayed where the administrators can grant the selected method access permissions to a user, user group, or all users. See: Creating Security Grants, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

7. (Optional) Users who have the Integration Repository Administrator role can generate SOAP services if the custom interfaces can be service enabled.

   This is achieved by first locating the custom interface, and then specifying the interaction pattern either at the interface level or the method level before clicking **Generate** (or **Generate WSDL**) in the selected custom interface details page. See: Generating Custom SOAP Web Services, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

8. (Optional) Users who have the Integration Repository Administrator role can deploy the web services from Oracle Integration Repository to the application server.

   To deploy generated SOAP services, the administrators must first select one authentication type (Username Token or SAML Token) for each selected service and then click **Deploy** in the selected interface details page. This deploys the generated service to the application server. See: Deploying and Undeploying Custom SOAP Web Services, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

   If the custom interfaces can be exposed as REST services, the administrators must enter a unique service alias for each selected custom interface before deploying the service. If the selected interface is an interface type of Java Bean Services, Application Module Services, or Business Service Object, the administrators need to specify HTTP methods for the service operations contained in the selected interface.

Note: Although Open Interface Tables and Open Interface Views can be exposed as REST services, custom Open Interfaces are not supported in this release.

See: Deploying Custom REST Web Services, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

Custom Integration Interface Annotation Example

The key essence of successfully creating custom integration interfaces relies on properly explanation of the new interface feature or definition. Once a custom interface definition of a specific interface type is created, an integration developer must properly annotate the custom file based on the Integration Repository annotation standards so that the interface file of a specific interface type can be displayed with appropriate description from the browser interface.

For example, the integration developer can create a Supplier Ship and Debit Request custom interface using PL/SQL API. This custom PL/SQL API package specification file (zz_sdrequest_s.pls) can be as follows:

```
set verify off
whenever sqlerror exit failure rollback;
WHENEVER OSERROR EXIT FAILURE ROLLBACK;

create or replace package ZZ_SDREQUEST as
/* $Header: zz_sdrequest_s.pls $ */

-- Custom procedure to create single supplier ship and debit request

procedure ZZ_CREATE_SDREQUEST (
CP_API_VERSION_NUMBER  IN    NUMBER,
CP_INIT_MSG_LIST  IN    VARCHAR2 := FND_API.G_FALSE,
CP_COMMIT  IN    VARCHAR2 := FND_API.G_FALSE,
CP_VALIDATION_LEVEL  IN    NUMBER := FND_API.G_VALID_LEVEL_FULL,
CX_RETURN_STATUS  OUT    VARCHAR2,
CX_MSG_COUNT  OUT    NUMBER,
CX_MSG_DATA  OUT    VARCHAR2,
CP_SDR_HDR_REC  IN    OZF_SD_REQUEST_PUB.SDR_HDR_REC_TYPE,
CP_SDR_LINES_REC  IN    OZF_SD_REQUEST_PUB.SDR_lines_rec_type,
CP_SDR_CUST_REC  IN    OZF_SD_REQUEST_PUB.SDR_cust_rec_type,
CP_SDR_BILLTO_REC  IN    OZF_SD_REQUEST_PUB.SDR_cust_rec_type,
CX_REQUEST_HEADER_ID  OUT    NUMBER
)
;
end ZZ_SDREQUEST;

/
commit;
exit;
```

Based on the PL/SQL API annotation standards, the integration developer must annotate the Supplier Ship and Debit Request custom package specification file by adding the annotation information specifically in the following places:

- Annotate the PL/SQL API package specification

- Annotate the PL/SQL procedure

  The annotations for the procedure should be placed between the definition and ';'.

Please note that you only need to annotate the custom package specification file, but not the package body file.

How to annotate custom interfaces for the interface types supported by Oracle Integration Repository, see Integration Repository Annotation Standards, page A-1.

```
set verify off
whenever sqlerror exit failure rollback;
WHENEVER OSERROR EXIT FAILURE ROLLBACK;

create or replace package ZZ_SDREQUEST as
/* $Header: zz_sdrequest_s.pls $ */
/*#
* This custom PL/SQL package can be used to create supplier ship and
debit request for single product.
* @rep:scope public
* @rep:product OZF
* @rep:displayname Single ship and debit request
* @rep:category BUSINESS_ENTITY OZF_SSD_REQUEST
*/

-- Custom procedure to create single supplier ship and debit request

procedure ZZ_CREATE_SDREQUEST (
CP_API_VERSION_NUMBER  IN    NUMBER,
CP_INIT_MSG_LIST  IN    VARCHAR2 := FND_API.G_FALSE,
CP_COMMIT  IN    VARCHAR2 := FND_API.G_FALSE,
CP_VALIDATION_LEVEL  IN    NUMBER := FND_API.G_VALID_LEVEL_FULL,
CX_RETURN_STATUS  OUT    VARCHAR2,
CX_MSG_COUNT  OUT    NUMBER,
CX_MSG_DATA  OUT    VARCHAR2,
CP_SDR_HDR_REC  IN    OZF_SD_REQUEST_PUB.SDR_HDR_REC_TYPE,
CP_SDR_LINES_REC  IN    OZF_SD_REQUEST_PUB.SDR_lines_rec_type,
CP_SDR_CUST_REC  IN    OZF_SD_REQUEST_PUB.SDR_cust_rec_type,
CP_SDR_BILLTO_REC  IN    OZF_SD_REQUEST_PUB.SDR_cust_rec_type,
CX_REQUEST_HEADER_ID  OUT    NUMBER
)
/*#
* Use this procedure to create single supplier ship and debit request
* @param CP_API_VERSION_NUMBER Version of the custom API
* @param CP_INIT_MSG_LIST Flag to initialize the message stack
* @param CP_COMMIT Indicates Flag to commit within the program
* @param CP_VALIDATION_LEVEL Indicates the level of the validation
* @param CX_RETURN_STATUS Indicates the status of the program
* @param CX_MSG_COUNT Provides the number of the messages returned by
the program
* @param CX_MSG_DATA Returns messages by the program
* @param CP_SDR_HDR_REC Contains details of the new Ship Debit Request
to be created
* @param CP_SDR_LINES_REC Contains the product line information for the
new Ship Debit Request
* @param CP_SDR_CUST_REC Contains the Customer information for the new
Ship Debit Request
* @param CP_SDR_BILLTO_REC Contains the Bill-to information for the new
Ship Debit Request
* @param CX_REQUEST_HEADER_ID Returns the id of the new Ship Debit
Request created
* @rep:displayname Create ship and debit request
* @rep:category BUSINESS_ENTITY OZF_SSD_REQUEST
* @rep:scope public
* @rep:lifecycle active
*/
;
end ZZ_SDREQUEST;

/
commit;
exit;
```

Generating and Uploading iLDT Files

Once annotated custom integration interface definitions are created, these annotated source files need to be validated against the annotation standards before they can be uploaded to Oracle Integration Repository. This validation is performed by executing the Integration Repository Parser (IREP Parser), a design time tool, to read the annotated files and then generate an Integration Repository loader file (iLDT) if no error occurred.

> **Note:** Please note that Integration Repository Parser does not support the integration interfaces registered under custom applications.
>
> It is currently tested and certified for Linux, Unix, Oracle Solaris on SPARC, HP-UX Itanium, HP-UX PA-RISC, IBM AIX on Power Systems and Windows.

Once an iLDT file is generated, an integration repository administrator can upload the generated file to Oracle Integration Repository where the custom interfaces can be exposed to all users.

How to set up the Integration Repository Parser, and use it to generate and upload the iLDT file, refer to:

- Setting Up and Using Integration Repository Parser, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*
- Generating ILDT Files, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*
- Uploading ILDT Files to Integration Repository, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*

Viewing Custom Interfaces and Performing Administrative Tasks

Searching and Viewing Custom Interfaces

Once annotated custom interface definitions are uploaded successfully, they are merged into the interface types they belong to and displayed together with Oracle interfaces from the Integration Repository browser window. To easily distinguish annotated custom interface definitions from Oracle interfaces, the Interface Source "Custom" is used to categorize those custom integration interfaces in contrast to Interface Source "Oracle" for Oracle interfaces.

To search for custom integration interfaces, you can use either one of the following ways:

- From the Interface List page, select 'Custom' from the Interface Source drop-down list along with a value for the Scope field to restrict the custom integration interfaces display.

- From the Search page, click **Show More Search Options** to select 'Custom' from the Interface Source drop-down list along with any interface type, product family, or scope if needed as the search criteria.

After executing the search, all matched custom integration interfaces will be displayed. For more information on how to search and view custom integration interfaces, see Searching Custom Integration Interfaces, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide* and Viewing Custom Integration Interfaces, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

Performing Administrative Tasks

Once custom integration interfaces are uploaded and displayed from the Integration Repository browser interface types, all the administrative tasks are the same for the native integration interfaces. These administrative tasks including creating security grants for newly created custom interfaces if needed, generating Web services, and deploying Web services. See Administering Custom Integration Interfaces and Services, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

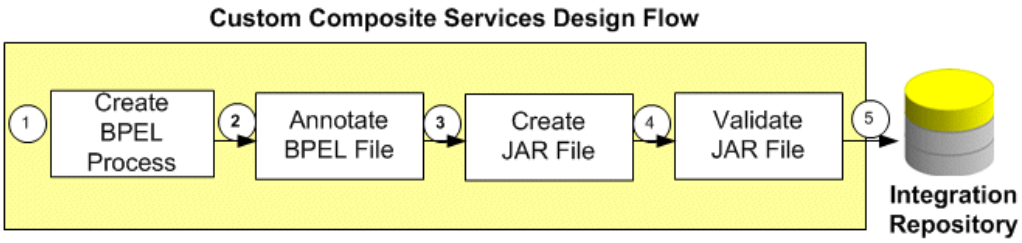
How to use custom integration interfaces as Web services to perform necessary transactions for your business needs, see a custom interface example described in the Using Custom Integration Interfaces as Web Services, page 11-24.

Creating Custom Integration Interfaces of Composite Services

Integration developers can create new composite services by orchestrating discrete Web services into meaningful business processes using BPEL language. With appropriate annotation specifically for the composite services and validation against the annotation standards, the validated JAR files of the composite service BPEL projects can be uploaded to the Integration Repository.

Creating Custom Composite Services

The following diagram illustrates the custom integration interface design flow for composite service - BPEL type:



1. A system integration developer orchestrates a composite service using a Web service composition language BPEL.

2. The integration developer annotates the composite service based on the Integration

Repository annotation standards specifically for the composite service interface type.

See: Creating and Annotating Custom Composite Services, page 11-11.

3. The integration developer creates a JAR file of the composite service BPEL project.

4. The integration repository administrator unzips the JAR file first and then validates the annotated custom interface definitions against the annotation standards specifically for composite services. This validation is performed by executing the Integration Repository Parser to read the annotated files and then generate an Integration Repository loader file (iLDT) if no error occurs.

5. An integration repository administrator uploads the generated iLDT file to Oracle Integration Repository through backend processing.

   See: Generating and Uploading iLDT Files, page 11-14.

   After the upload, you can search and view the uploaded custom interface from the Integration Repository user interface for verification.

Once custom integration interface definitions are uploaded and displayed from the Integration Repository browser, integration repository administrators and the integration developers can download the composite services for modification if needed. For information on how to download composite services, see Viewing and Downloading Custom Composite Services, page 11-15.

Custom Composite Service Annotation Example

As mentioned earlier that the key essence of successfully creating custom integration interfaces relies on properly explanation of the new interface feature or definition. Once a custom interface definition of a specific interface type is created, an integration developer must properly annotate the custom source file based on the Integration Repository annotation standards so that the interface file of a specific interface type can be displayed with appropriate description from the browser interface.

For example, a create invoice composite service - BPEL project is created. To annotate the composite service `*.bpel` file, you open the `*.bpel` file in text editor and place the annotation within the comments section in the beginning of the file as highlighted below:

```
////////////////////////////////////////////////////////////
Oracle JDeveloper BPEL Designer

  Created: Tue Oct 30 17:10:13 IST 2007
  Author:  <username>
  Purpose: Synchronous BPEL Process
 /*#
  * This is a bpel file for creating invoice.
  * @rep:scope public
  * @rep:displayname Create Invoice
  * @rep:lifecycle active
  * @rep:product PO
  * @rep:compatibility S
  * @rep:interface oracle.apps.po.CreateInvoice
  * @rep:category BUSINESS_ENTITY INVOICE
  */

////////////////////////////////////////////////////////////

-->
<process name="CreateInvoice">
   targetNamespace="http://xmlns.oracle.com/CreateInvoice"
     xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
     xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-
process/"
         xmlns:xp20="http://www.oracle.com/XSL/Transform/java/oracle.
tip.pc.services.functions.Xpath20"
         xmlns:ns4="http://xmlns.oracle.
com/pcbpel/adapter/file/ReadPayload/"
         xmlns:ldap="http://schemas.oracle.com/xpath/extension/ldap"
         xmlns:xsd="http://www.w3.org/2001/XMLSchema"
         xmlns:ns5="http://xmlns.oracle.com/bpel/workflow/xpath"
         xmlns:client="http://xmlns.oracle.com/CreateInvoice"
         xmlns:ns6="http://xmlns.oracle.
com/bpel/services/IdentityService/xpath"
         xmlns:ora="http://schemas.oracle.com/xpath/extension"
         xmlns:ns1="http://xmlns.oracle.
com/soaprovider/plsql/AR_INVOICE_API_PUB_2108/CREATE_SINGLE_INVOICE_1037
895/"
         xmlns:ns3="http://xmlns.oracle.
com/soaprovider/plsql/AR_INVOICE_API_PUB_2108/APPS/BPEL_CREATE_SINGLE_IN
VOICE_1037895/AR_INVOICE_API_PUB-24CREATE_INV/"
         xmlns:ns2="http://xmlns.oracle.com/pcbpel/adapter/appscontext/"
         xmlns:bpelx="http://schemas.oracle.com/bpel/extension"
   xmlns:orcl="http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.
services.functions.ExtFunc">

  <!--
////////////////////////////////////////////////////////////////////
PARTNERLINKS
      List of services participating in this BPEL process
////////////////////////////////////////////////////////////////////
-->
<partnerLinks>
 <!--
  The 'client' role represents the requester of this service. It is
      used for callback. The location and correlation information
associated
      with the client role are automatically set using WS-Addressing.
    -->
    <partnerLink name="client" partnerLinkType="client:CreateInvoice"
                 myRole="CreateInvoiceProvider"/>
    <partnerLink name="CREATE_SINGLE_INVOICE_1037895"
                 partnerRole="CREATE_SINGLE_INVOICE_1037895_ptt_Role"
                 partnerLinkType="ns1:
CREATE_SINGLE_INVOICE_1037895_ptt_PL"/>
```

```
<parnterLink name="ReadPayload" partnerRole="SynchRead_role"
                partnerLinkType="ns4:SynchRead_plt"/>
</partnerLinks>
<!--
//////////////////////////////////////////////////////////////////////
VARIABLES
      List of messages and XML documents used within this BPEL process
//////////////////////////////////////////////////////////////////////
-->
<variables>
<!--Reference to the message passed as input during initiation-->
  <variable name="inputVariable"
              messageType="client:CreateInvoiceRequestMessage"/>
<!--Reference to the message that will be returned to the requester-->
    <variable name="outputVariable"
              messageType="client:CreateInvoiceResponseMessage"/>
    <variable name="
Invoke_1_CREATE_SINGLE_INVOICE_1037895_InputVariable"
              messageType="ns1:Request"/>
    <variable name="
Invoke_1_CREATE_SINGLE_INVOICE_1037895_OutputVariable"
              messageType="ns1:Response"/>
    <variable name="Invoke_2_SynchRead_InputVariable"
              messageType="ns4:Empty_msg"/>
    <variable name="Invoke_2_SynchRead_OutputVariable"
              messageType="ns4:InputParameters_msg"/>
</variables>
<!--
//////////////////////////////////////////////////////////////////////
ORCHESTRATION LOGIC
     Set of activities coordinating the flow of messages across the
     services integrated within this business process
//////////////////////////////////////////////////////////////////////
-->
<sequence name="main">
    <!--Receive input from requestor. (Note: This maps to operation
defined in CreateInvoice.wsdl)-->
  <receive name="receiveInput" partnerLink="client"
             portType="client:CreateInvoice" operation="process"
             variable="inputVariable" createInstance="yes"/>
    <!--Generate reply to synchronous request-->
    <assign name="SetHeader">
      <copy>
        <from expression="''operations'">
        <to variable="
Invoke_1_CREATE_SINGLE_INVOICE_1037895_InputVariable"
            part="header"
            query="/ns1:SOAHeader/ns2:ProcedureHeaderType/ns2:Username"
/>
      </copy>
      <copy>
        <from expression="''Receivables, Vision Operations (USA)'">
        <to variable="
Invoke_1_CREATE_SINGLE_INVOICE_1037895_InputVariable"
            part="header"
            query="/ns1:SOAHeader/ns2:ProcedureHeaderType/ns2:
Responsibility"/>
      </copy>
      <copy>
        <from expression="''204'">
        <to variable="
Invoke_1_CREATE_SINGLE_INVOICE_1037895_InputVariable"
            part="header"
            query="/ns1:SOAHeader/ns2:ProcedureHeaderType/ns2:ORG_ID"/>
      </copy>
      <copy>
```

```
<from expression="''Receivables, Vision Operations (USA)'">
        <to variable="
Invoke_1_CREATE_SINGLE_INVOICE_1037895_InputVariable"
            part="header"
            query="/ns1:SOAHeader/ns1:SecurityHeader/ns1:
ResponsibilityName"/>
      </copy>
   </assign>
   <invoke name="InvokeReadPayload" partnerLink="ReadPayload"
            portType="ns4:SynchRead_ptt" operation="SynchRead"
            inputVariable="Invoke_2_SynchRead_InputVariable"
            outputVariable="Invoke_2_SynchRead_OutputVariable"/>
   <assign name="SetPayload">
      <copy>
        <from variable="Invoke_2_SynchRead_OutputVariable"
              part="InputParameters" query="/ns3:InputParameters"/>
        Invoke_1_CREATE_SINGLE_INVOICE_1037895_InputVariable"
        part="body" query="/ns1:SOARequest/ns3:InputParameters"/>
      </copy>
   </assign>
   <assign name="SetDate">
      <copy>
        <from expression="xp20:current-date()">
        <to to variable="
Invoke_1_CREATE_SINGLE_INVOICE_1037895_InputVariable"
            part="body"
            query="/ns1:SOARequest/ns3:InputParameters/ns3:
P_TRX_HEADER_TBL/ns3:P_TRX_HEADER_TBL_ITEM/ns3:TRX_DATE"/>
      </copy>
   </assign>
   <invoke name="Invoke_1" partnerLink="CREATE_SINGLE_INVOICE_1037895"
            portType="ns1:CREATE_SINGLE_INVOICE_1037895_ptt"
            operation="CREATE_SINGLE_INVOICE_1037895"
            inputVariable="
Invoke_1_CREATE_SINGLE_INVOICE_1037895_InputVariable"
            outputVariable="
Invoke_1_CREATE_SINGLE_INVOICE_1037895_OutputVariable"/>
   <assign name="AssignResult">
      <copy>
        <from variable="
Invoke_1_CREATE_SINGLE_INVOICE_1037895_OutputVariable"
              part="body"
              query="/ns1:SOAResponse/ns3:OutputParameters/ns3:
X_MSG_DATA"/>
        <to variable="outputVariable" part="payload"
            query="/client:CreateInvoiceProcessResponse/client:result"/>
      </copy>
   </assign>
   <reply name="replyOutput" partnerLink="client"
           portType="client:CreateInvoice" operation="process"
           variable="outputVariable"/>
 </sequence>
</process>
```

For more information on how to annotate composite service BPEL type, see Composite Service - BPEL Annotations, page A-110.

Generating and Uploading iLDT Files

Once annotated custom composite services are created, these annotated source files need to be validated against the annotation standards specifically for composite service - BPEL type before they can be uploaded to Oracle Integration Repository. This validation is performed by executing the Integration Repository Parser (IREP Parser), a

design time tool, to read the annotated files and then generate an Integration Repository loader file (iLDT) if no error occurred.

> **Note:** Please note that Integration Repository Parser does not support the integration interfaces registered under custom applications.
>
> It is currently tested and certified for Linux, Unix, Oracle Solaris on SPARC, HP-UX Itanium, HP-UX PA-RISC, IBM AIX on Power Systems and Windows.

Once an iLDT file is generated, an integration repository administrator can upload the generated file to Oracle Integration Repository where the custom interfaces can be exposed to all users.

How to set up the Integration Repository Parser, and use it to generate and upload the iLDT file, see:

- Setting Up and Using Integration Repository Parser, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*
- Generating ILDT Files, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*
- Uploading ILDT Files to Integration Repository, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*

Viewing and Downloading Custom Composite Services

Once annotated custom composite service definitions are uploaded successfully, they are merged into the Composite Service BPEL type and displayed together with Oracle interfaces from the Integration Repository browser window. To easily distinguish annotated custom composite services from Oracle interfaces, the Interface Source "Custom" is used to categorize those custom interfaces in contrast to Interface Source "Oracle" for Oracle interfaces.

To search for custom composite services, from the Search page, click **Show More Search Options** to expand the search criteria. Select 'Custom' from the Interface Source drop-down list along with 'Composite Service' interface type, product family, or scope if needed as the search criteria.

After executing the search, all matched custom composite services will be displayed.

Downloading Custom Composite Services

Similar to downloading native packaged composite services, the integration repository administrators and the integration developers can click **Download Service** in the composite service interface details page to download the relevant custom composite files aggregated in a .JAR file to your local directory.

For more information on how to search and download custom composite services, see

Downloading Composite Services, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

Creating Custom Business Events Using Workflow XML Loader

Oracle E-Business Suite Integrated SOA Gateway allows you to create custom business events in the Business Event System and then download the events that you have created, annotate the event source codes, validate the files, and then upload the files back to the event system using Workflow XML Loader.

The Workflow XML Loader is a command line utility that lets you upload and download XML definitions for Business Event System objects between a database and a flat file. When you download Business Event System object definitions from a database, Oracle Workflow saves the definitions as an XML file. When you upload object definitions to a database, Oracle Workflow loads the definitions from the source XML file into the Business Event System tables in the database, creating new definitions or updating existing definitions as necessary.

XML files uploaded or downloaded by the Workflow XML Loader should have the extension `.wfx` to identify them as Workflow object XML definitions.

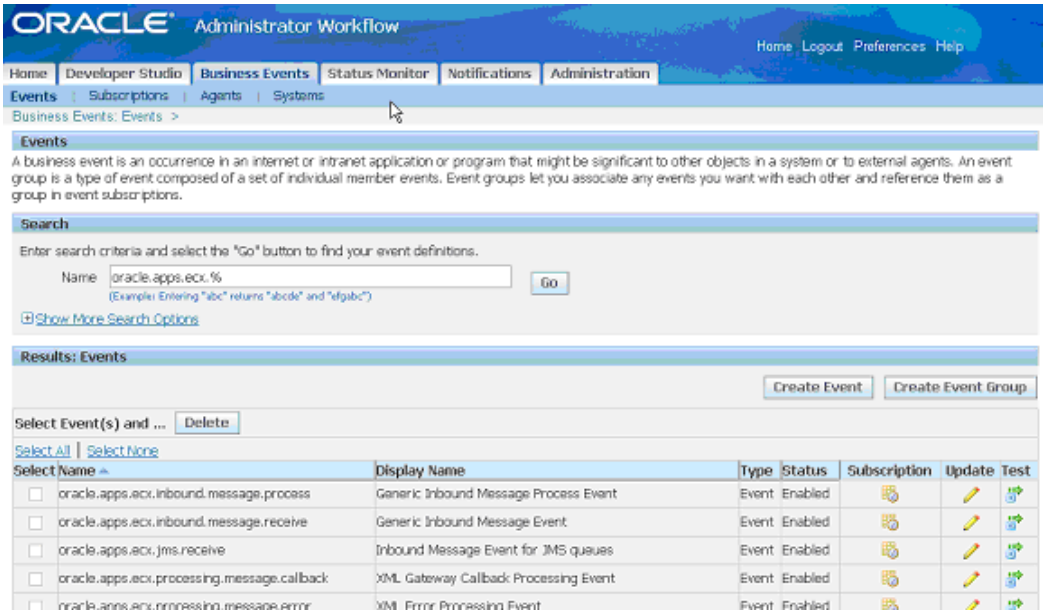
Use the following steps to create custom business events:

1. Locate and Download Business Events, page 11-16
2. Annotate the XML Definition File, page 11-19
3. Validate the Annotated Source File Using Integration Repository Parser, page 11-21
4. Upload Annotated File to the Database, page 11-22
5. Upload iLDT Files to Integration Repository, page 11-23

Step 1: Locating and Downloading Business Events

After creating custom business events in the Oracle Workflow Business Event System, first locate them and then download them using Workflow XML Loader.

Business Events Page



To download XML definitions for Business Event System objects between a database and a flat file, run the Workflow XML Loader by running Java against `oracle.apps.fnd.wf.WFXload` with the following command syntax:

```
jre oracle.apps.fnd.wf.WFXload -d{e} <user> <password> <connect string>
<protocol> <language> <xml file> <object> {<key>} {<OWNER_TAG>}
{<owner>}
```

For example, you can download either a single event or a group of events:

- Use the following command to download a single business event, such as **`wf`**`demoe.wfx`. In the filename, first two or three chars refers to the product and the last character 'e' refers to Event.

  ```
  java oracle.apps.fnd.wf.WFXLoad -d apps_read_only password hostdb:
  xxxxx:sidxxx thin US wfdemoe.wfx EVENTS abc.apps.wf.bes.demo.event
  ```

- Use the following command to download a group of business events with wildcard:

  ```
  java oracle.apps.fnd.wf.WFXLoad -d apps_read_only password hostdb:
  xxxxx:sidxxx thin US wfdemoe.wfx EVENTS abc.apps.wf.bes.%
  ```

After successfully downloading the event XML definitions, open the `.wfx` file in any text editor. You will find the content of a `wfdemoe.wfx` file, for example, containing one event shown as follows:

```
<?xml version = '1.0' encoding = 'UTF-8'?>
...

<oracle.apps.wf.event.all.sync><ExternalElement>
<OraTranslatibility>
<XlatElement Name="WF_EVENTS">
<XladID>
<Key>NAME</Key>
</XladID>
<XlatElement Name="DISPLAY_EVENTS" MaxLen="80" Expansion="50"/>
<XladID>
<Key Type="CONSTANT">DISPLAY_EVENTS</Key>
</XladID>
<XlatElement Name="DESCRIPTION" MaxLen="2000" Expansion="50"/>
<XladID>
<Key Type="CONSTANT">DESCRIPTION</Key>
</XladID>
</XlatElement>
</OraTranslatibility>
</ExternalElement>
<WF_TABLE_DATA>
  <WF_EVENTS>
   <VERSION>1.0</VERSION>
  <GUID>#NEW</GUID>
  <NAME>abc.apps.wf.demo.event</NAME>
  <TYPE>EVENT</TYPE>
  <STATUS>ENABLED</STATUS>
  <GENERATE_FUNCTION/>
  <OWNER_MAME>Oracle Workflow</OWNER_MAME>
  <OWNER_TAG>FMD</OWNER_TAG>
  <CUSTOMIZATION_LEVEL>U</CUSTOMIZATION_LEVEL>
  <LICENSED_FLAG>Y</LICENSED_FLAG>
  <JAVA_GENERATE_FUNC/>
  <DISPLAY_NAME>Demo Business Event</DISPLAY_NAME>
  <DESCRIPTION>Business event created for annotation demo.</DESCRIPTION>
  <IREP_ANNOTATION>/*#
* Business event created for annotation demo.
*
* @rep:scope public
* @rep:displayname Demo Business Event
* @rep:product FND
* @rep:category BUSINESS_ENTITY
*/
</IREP_ANNOTATION>
  </WF_EVENTS>
</WF_TABLE_DATA>
</oracle.apps.wf.event.all.sync>
```

The Workflow XML Loader automatically creates a template for integration repository annotation as highlighted in bold between `<IREP_ANNOTATION>` and `</IREP_ANNOTATION>`. This is where appropriate annotations need to be placed or modified for a business event based on the business event annotation standards.

To download business events XML definitions:

1. Log on to Oracle Workflow page with the Workflow Administrator Web Applications responsibility. Select the Business Events link from the Navigator to open the Events page.

2. Enter search criteria in the Search region to locate your business events.

3. Change your directory to the same environment where your application is running.

For example, if your application is running on `seed100`, then change your directory to `seed100` where your business events exist.

```
/slot/ems3404/appmgr/apps/apps_st/appl
. ./APPSseed100.env
```

4. Download the events from the database using `oracle.apps.fnd.wf.WFXload` with the following syntax:

```
jre oracle.apps.fnd.wf.WFXload -d{e} <user> <password> <connect
string> <protocol> <language> <xml file> <object> {<key>}
{<OWNER_TAG>} {<owner>}
```

5. Open the `.wfx` file in any text editor and notice that one business event has been placed there.

Step 2: Annotating an XML Definition

After successfully downloading the XML definition file from a database, you should open the `.wfx` file containing one business event in any text editor and modify the annotation appropriately based on Integration Repository business event annotation standards.

The appropriate annotation includes:

- Enter meaningful description.
- Enter conditions under which the business event is raised.
- Enter UI action that invokes the business event if applicable.
- Verify scope. By default, the Workflow XML Loader annotates scope as 'public'.
- Verify display name. By default, the Workflow XML Loader uses the same display name as that mentioned in business event definition.
- Verify product. By default, the Workflow XML Loader uses Owner Tag as the Application Short Name.

  Make sure that the Owner Tag corresponds to Application Short Name in FND_APPLICATION. Owner Name typically corresponds to Application Name but if your product is part of a larger application, you may enter an appropriate name in Owner Name.
- Enter BUSINESS_ENTITY code that your respective business event belongs to.
- Enter additional annotation properties if needed.

Please note that the IREP properties should not be blank. For example, the Workflow XML Loader only adds the template for Business Entity as rep:category BUSINESS_ENTITY, page A-128, but you should add an appropriate business entity to

which the event belongs. Similarly, other `@rep` properties cannot be left blank either.

The following is a sample business event annotation for Oracle Workflow:

```
* Business Event created to demonstrate using WFXLoad to annotate
Business Events.
*
* @rep:scope internal
* @rep:displayname Demo Business Event
* @rep:product OWF
* @rep:lifecycle active
* @rep:category BUSINESS_ENTITY WF_EVENT
*/
```

Important: If you decide not to annotate or publish the event in Oracle Integration Repository, you should remove the annotation only but leave the following tags unchanged. Presence of these tags is an indication that the event was reviewed for annotation.

```
<IREP_ANNOTATION/>
```

or

```
<IREP_ANNOTATION></IREP_ANNOTATION>
```

If the Loader sees these empty tags, it interprets that the business event was reviewed for annotation and it does not need to be published to the Integration Repository. Next time, when the user downloads these events, the Loader will insert empty IREP_ANNOTATION tags as shown in the following example.

However, if you remove the entire IREP_ANNOTATION tags for the business event and upload it, then on subsequent download the Loader will insert partially filled annotation template for the business event.

```
<WF_TABLE_DATA>
  <WF_EVENTS>
   <VERSION>1.0</VERSION>
  <GUID>#NEW</GUID>
  <NAME>oracle.apps.wf.demo.event.noannotate</NAME>
  <TYPE>EVENT</TYPE>
  <STATUS>ENABLED</STATUS>
  <GENERATE_FUNCTION/>
  <OWNER_MAME>Oracle Workflow</OWNER_MAME>
  <OWNER_TAG>FMD</OWNER_TAG>
  <CUSTOMIZATION_LEVEL>U</CUSTOMIZATION_LEVEL>
  <LICENSED_FLAG>Y</LICENSED_FLAG>
  <JAVA_GENERATE_FUNC/>
  <DISPLAY_NAME>Demo Business Event with no
annotation</DISPLAY_NAME>
  <DESCRIPTION>Business second event created for
annotation demo.</DESCRIPTION><IREP_ANNOTATION>/*#
* Business event created for annotation demo.
*
* @rep:scope public
* @rep:displayname Demo Business Event
* @rep:product FND
* @rep:category BUSINESS_ENTITY
*/
</IREP_ANNOTATION>
  </WF_EVENTS>
</WF_TABLE_DATA>
```

For more information on Integration Repository Business Event Annotation Standards, see Business Event Annotations, page A-35.

Step 3: Validating the Annotated Source File Using Integration Repository Parser

Integration Repository Parser is a standalone design time tool. It can be executed to validate the annotated custom interface definitions against the annotation standards and to generate an iLDT file if no error occurs.

After annotating the XML definition for a business event, execute the standalone Integration Repository Parser (IREP Parser) using the following command syntax to validate whether the annotation in `.wfx` file is valid:

Command Syntax:

```
$IAS_ORACLE_HOME/perl/bin/perl $FND_TOP/bin/irep_parser.pl -g -v
-username=<a fnd username> <product>:<relative path from product
top>:<fileName>:<version>=<Complete File Path, if not in currect
directory>
```

For example:

```
$IAS_ORACLE_HOME/perl/bin/perl $FND_TOP/bin/irep_parser.pl -g -v
-username=sysadmin owf:patch/115/xml/US:wfdemoe.wfx:12.0=.
/wfdemoe.wfx
```

While executing the parser, pay attention to any error messages on the console. Typically these errors would be due to incorrect annotation or some syntax errors in the annotated file. Ensure that the annotations are correct and the file has proper syntax.

If no error occurs in the annotated interface file, an iLDT (`*.ildt`) file would be

generated. An integration repository administrator needs to upload the generated iLDT file to the Integration Repository where the custom business events can be exposed to all users. See Step 5: Uploading iLDT Files to Integration Repository, page 11-23.

Integration Repository Parser (irep_parser.pl)

The `irep_parser` is a design time tool. It reads interface annotation documentation in program source files and validates it according to its file type. If the `-generate` flag is supplied (and other conditions met), then it will generate iLDT files. Any validation errors will be reported, usually along with file name and line number, like the result of `grep -n`.

Additionally, it can handle almost all types of application source files. While validating the annotated files against the annotation standards of supported interface types, if files that do not match will be ignored.

The parser will return an exit value of 0 if no errors occurred during processing. Otherwise, it will return a count of the number of files that had errors. Files with incomplete information for generation (class resolution) are considered errors only if the `-generate` flag is used.

However, before executing the Integration Repository Parser, you need to install `perl` modules and apply necessary patches. For setup information, see Setting Up and Using Integration Repository Parser, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

For information on the Integration Repository Parser (irep_parser.pl) usage details including supported file types and options, files specifications, and environment, see Integration Repository Parser (irep_parser.pl) Usage Details, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

Step 4: Uploading Annotated File Back to a Database

After validating the annotated source file `.wfx`, upload the file back to the database where you downloaded it earlier so that the annotated file can be stored in the appropriate tables in business event system for future references.

> **Note:** To view custom business events through the Integration Repository browser window, an integration repository administrator needs to upload the generated iLDT files to the Integration Repository. For information on uploading iLDT files, see Step 5: Uploading iLDT Files to Integration Repository, page 11-23.

The Workflow XML Loader lets you upload business event system XML definitions in either normal upload mode (`-u`) or force upload mode (`-uf`):

- Normal upload mode (`-u`): If you created an event with a customization level of Core or Limit, the Workflow XML Loader will be able to update IREP_ANNOTATION into the Business Event System `WF_EVENTS` table in the database. This normal mode will not make any updates to events or subscriptions

with a customization level of User.

Use the following command to upload the annotated `.wfx` file back to a database:

```
java oracle.apps.fnd.wf.WFXLoad -u apps_read_only password
hostdb:xxxxx:sidxxx thin US wfdemoe.wfx
```

- Force upload mode (`-uf`): The Workflow XML Loader loads the object definitions from the source XML file into the Business Event System tables in the database and overwrites any existing definitions, even for events or subscriptions with a customization level of User.

  Therefore, if you created an event with a customization level of User, use the following force upload option to make sure the IREP_ANNOTATION can be uploaded back into the database.

  ```
  java oracle.apps.fnd.wf.WFXLoad -uf apps_read_only password
  hostdb:xxxxx:sidxxx thin US wfdemoe.wfx
  ```

For more information on how to use Workflow XML Loader, see Using the Workflow XML Loader, *Oracle Workflow Administrator's Guide*.

Step 5: Uploading ILDT Files to Integration Repository

After the validation using the Integration Repository Parser, an iLDT file will be generated if no error occurs during the iLDT generation. In order for users to view the custom business events through the Integration Repository, an integration repository administrator needs to manually upload the generated iLDT file to the Integration Repository using FNDLOAD command.

```
$FND_TOP/bin/FNDLOAD <db_connect> 0 Y UPLOAD
$fnd/patch/115/import/wfirep.lct <ildt file>
```

For example, `FND_TOP/bin/FNDLOAD apps @instance_name 0 Y UPLOAD $FND_TOP/patch/115/import/wfirep.lct SOAIS_pls.ildt`

`ORACLE Password: password`

For detailed information on how to upload the iLDT files, see Uploading ILDT Files to Integration Repository, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

Viewing Custom Interfaces and Performing Administrative Tasks

Searching and Viewing Custom Interfaces

Annotated custom interface definitions, once they are uploaded successfully, are merged into the interface types they belong to and displayed together with Oracle interfaces from the Integration Repository browser window. To easily distinguish annotated custom interface definitions from Oracle interfaces, the Interface Source "Custom" is used to categorize those custom integration interfaces in contrast to Interface Source "Oracle" for Oracle interfaces.

To search for custom integration interfaces, you can use either one of the following ways:

- From the Interface List page, select 'Custom' from the Interface Source drop-down list along with a value for the Scope field to restrict the custom integration interfaces display.

- From the Search page, click **Show More Search Options** to select 'Custom' from the Interface Source drop-down list along with any interface type (such as 'Business Event'), product family, or scope if needed as the search criteria.

After executing the search, all matched custom integration interfaces will be displayed. For more information on how to search and view custom integration interfaces, see Searching Custom Integration Interfaces, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide* and Viewing Custom Integration Interfaces, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

Performing Administrative Tasks

Once custom business events are uploaded and displayed from the Integration Repository browser interface types, all the administrative tasks are the same for the native interfaces. These administrative tasks including creating security grants for newly created custom events if needed, and subscribing to custom business events. See Administering Custom Integration Interfaces and Services, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

How to use custom integration interfaces as Web services to perform necessary transactions for your business needs, see a custom interface example described in the Using Custom Integration Interfaces as Web Services, page 11-24.

Using Custom Integration Interfaces as Web Services

Overview

With appropriate annotation and validation, a custom integration interface can be created or built around a business entity for the interface type that Oracle Integration Repository supports. If the interface type that the custom interface belongs to can be service enabled, you can use the custom interface as a Web service to update or retrieve data from Oracle E-Business Suite or perform other business transactions over the Web.

For example, an integration developer can create a new or customized interface for Supplier Ship and Debit Request business entity using a PL/SQL API. Once the interface is uploaded to Oracle Integration Repository, it will be displayed under the PL/SQL API interface type from the Integration Repository browser. To differentiate the custom interfaces from Oracle native packaged ones, all custom integration interfaces have Interface Source 'Custom' in contrast to Oracle interfaces with Interface Source 'Oracle' when you view them from the repository.

To better understand how to use deployed custom interfaces as Web services in fulfilling your business needs, detailed design-time and run-time tasks in creating and deploying a BPEL process are discussed in this section. For the example described in the

following sections, we use Oracle JDeveloper 10.1.3.3.0 as a design-time tool to create the BPEL process and use Oracle SOA Suite BPEL server 10.1.3.3.0 for the process deployment.

Using Custom Interface WSDL in Creating a BPEL Process at Design Time

BPEL Process Scenario

Take a custom PL/SQL API `ZZ_SDREQUEST` as an example to explain the BPEL process creation.

When the request of creating a supplier ship and debit request is received, the request information including payload and request number will be read and passed to create a supplier ship and debit request. Once the supplier ship and debit request for a product is created, the request number will then be returned to the requestor.

After deploying the BPEL process, you should find a supplier ship and debit request is created in the Oracle E-Business Suite. The request number should be the same as the payload input value.

Prerequisites to Create a BPEL Process Using a Custom Web Service

Before performing design-time tasks for concurrent programs, ensure the following tasks are in place:

- An integration repository administrator needs to locate the custom interface and then create security grants so that the right person can access the interface.
- An integration repository administrator needs to successfully deploy the generated custom Web service to the application server.
- An integration developer needs to locate and record the deployed WSDL URL for the custom interface exposed as a Web service.

Creating Security Grants on the Custom Interface

To be able to verify and use this custom interface, the administrator will first locate the custom interface (with 'Custom' interface source) from the repository, and then create security grants for the custom interface so that users with appropriate privileges can execute the interface and access the application for secured transactions.

For example, the administrator can grant the custom API access privilege to a user who has Oracle Trade Management responsibility. After the execution of this custom API, the authorized user can log in to Oracle Trade Management and verify the supplier and debit request creation details.

How to create security grants, see Creating Grants, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

Deploying the WSDL URL for the Custom Interface

An integration repository administrator must perform the following steps before letting integration developers use the deployed WSDL in creating a BPEL process:

1. To generate a Web service, locate the interface definition first (such as a custom PL/SQL interface ZZ_SDREQUEST) and click **Generate** in the SOAP Web Service tab of the interface details page.

   For detailed instruction on how to generate a Web service, see Generating SOAP Web Services, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

2. To deploy a generated Web service, select one authentication type and click **Deploy** in the SOAP Web Service tab to deploy the service.

   Once the service is successfully deployed, the selected authentication type will be displayed along with 'Deployed' Web Service Status. For more information on securing Web services with authentication type, see Managing Web Service Security, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

   For detailed instruction on how to deploy a Web service, see Deploying, Undeploying, and Redeploying SOAP Web Services, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

Searching and Recording WSDL URL

Apart from the required tasks performed by the administrators, an integration developer also needs to log on to the system to locate and record the deployed Web service WSDL URL for the custom interface that needs to be orchestrated into a meaningful business process in Oracle JDeveloper using BPEL language.

This WSDL information will be used later in creating a partner link for the custom interface exposed as a Web service during the BPEL process creation at design time.

For information on how to search for an interface and review the interface details, see Searching and Viewing Integration Interfaces, page 2-2.

BPEL Process Creation Flow

Based on the supplier and debit request creation scenario, the following design-time tasks are discussed in this chapter:

1. Create a new BPEL project, page 11-27

   Use this step to create a new BPEL project called ZZ_CreateSingle_ShipDebitRequest.bpel using an Synchronous BPEL Process template. This automatically creates two dummy activities - Receive and Reply - to receive input from a third party application and to reply output of the BPEL process back to the request application.

2. Create a Partner Link, page 11-28

   Use this step to create an invoice in Oracle E-Business Suite by using the Single Ship and Debit Request custom API ZZ_SDREQUEST exposed as Web service.

3. Add a Partner Link for File Adapter, page 11-31

   Use this step to synchronous read input data details passed from the first Assign

activity to create supplier ship and debit request.

4. Add Invoke activities, page 11-35

   Use this step to configure two Invoke activities in order to:

   - Point to the File Adapter to synchronous read input data details that is passed from the first Assign activity.
   - Point to the `ZZ_SDREQUEST` partner link to initiate the supplier ship and debit request creation with payload and request number details received from the Assign activities.

5. Add Assign activities, page 11-37

   Use this step to configure Assign activities in order to pass application context header variables, payload information and request number to appropriate Invoke activities to facilitate the single supplier ship and debit request creation. At the end, pass the request number to the request application through the dummy Reply activity.

For general information and basic concept of a BPEL process, see Understanding BPEL Business Processes, page D-1 and the *Oracle BPEL Process Manager Developer's Guide* for details.

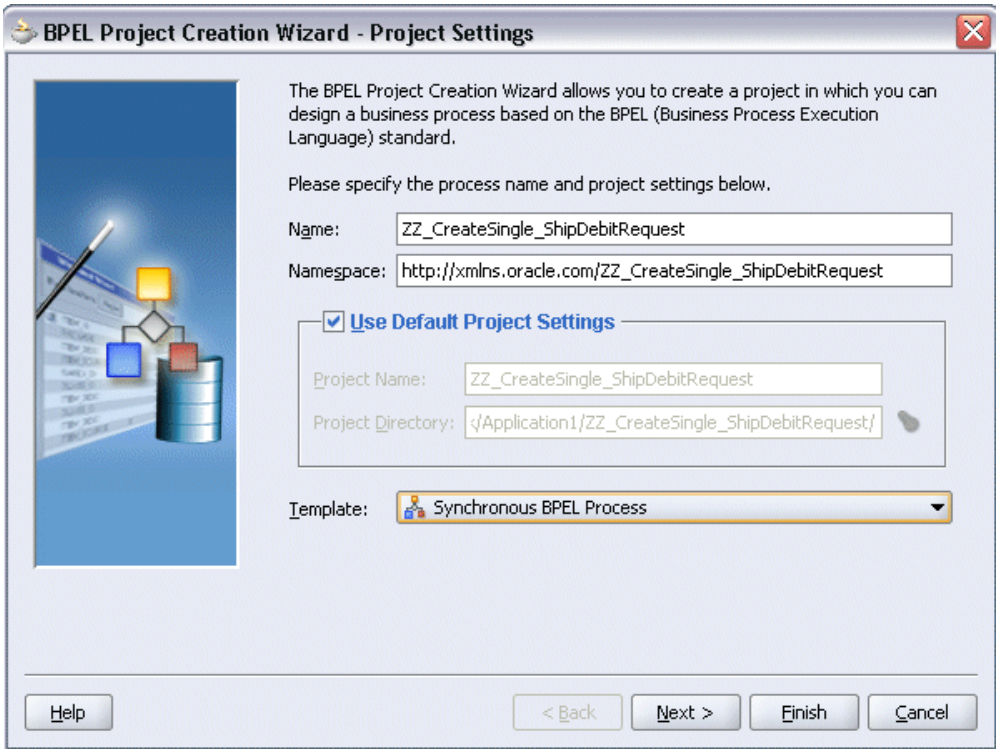
Creating a New BPEL Project

Use this step to create a new BPEL project that will contain various BPEL process activities.

To create a new BPEL project:

1. Open JDeveloper BPEL Designer.
2. From the **File** menu, select **New**. The New Gallery dialog box appears.
3. Select **All Items** from the **Filter By** box. This produces a list of available categories.
4. Expand the **General** node and select **Projects**.
5. Select **BPEL Process Project** from the **Items** group.
6. Click **OK**. The BPEL Process Project dialog box appears.

BPEL Project Creation Wizard - Project Settings Dialog



7. In the **Name** field, enter a descriptive name such as `ZZ_CreateSingle_ShipDebitRequest`.

8. From the Template list, select **Synchronous BPEL Process** and then select **Use Default Project Settings**.

9. Use the default input and output schema elements in the Input/Output Elements dialog box.

10. Click **Finish**.

    A new synchronous BPEL process is created with the Receive and Reply activities. The required source files including bpel.xml, using the name you specified (for example, `ZZ_CreateSingle_ShipDebitRequest.bpel`) are also generated.

Creating a Partner Link for the Web Service

Use this step to create a Partner Link called `ZZ_CreateSD_Request`.

To create a partner link for Single Ship and Debit Request Web service:

1. In JDeveloper BPEL Designer, drag and drop the **PartnerLink** service from the Component Palette into the Partner Link border area of the process diagram. The

Service Name dialog box appears.

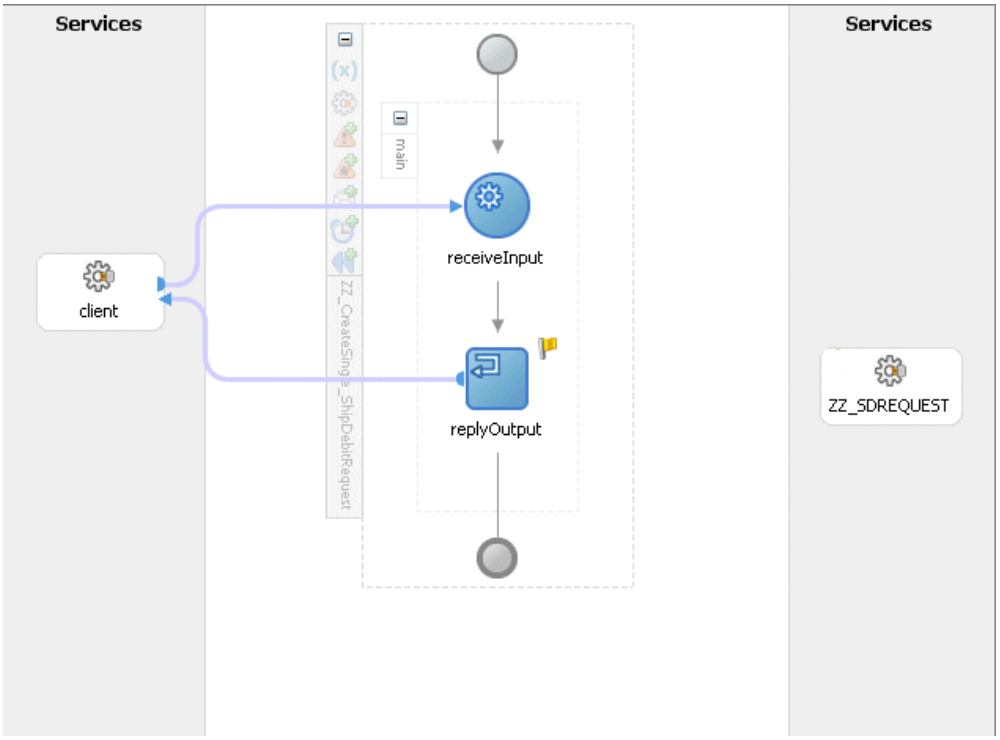
2. Copy the WSDL URL corresponding to the custom service, `ZZ_SDREQUEST`, that you recorded earlier in the WSDL File field.

3. A Partner Link Type message dialog box appears asking whether you want the system to create a new WSDL file that will by default create partner link types for you.

   Click **Yes** to have the Partner Name value populated automatically.

   The partner link is created with the required WSDL settings, and is represented in the BPEL project by a new icon in the border area of the process diagram.

BPEL Process Diagram with Partner Link

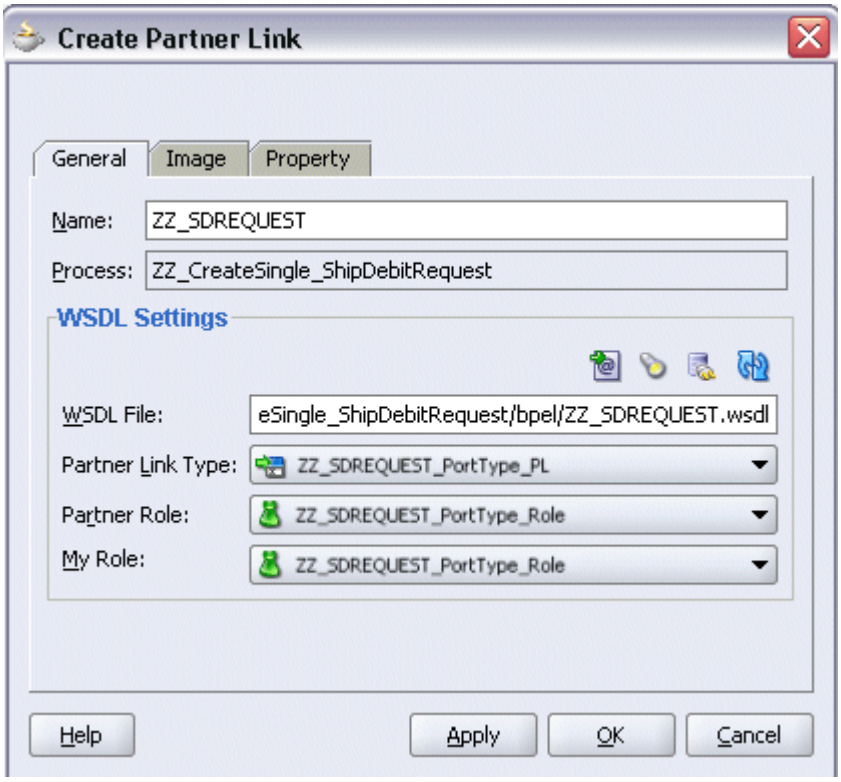


4. You can optionally change the default partner link name by double-clicking the icon to open the Edit Partner Link window if you like.

   Select the Partner Role value from the drop-down list.

   Click **Apply**.

Create Partner Link Dialog



5. Select the Property tab and click the **Create Property** icon to select the following properties from the property name drop-down list in order to pass the security header along with the SOAP request:

   - wsseUsername

     Specify the username such as `trademgr` to be passed in the Property Value box.

   - wssePassword

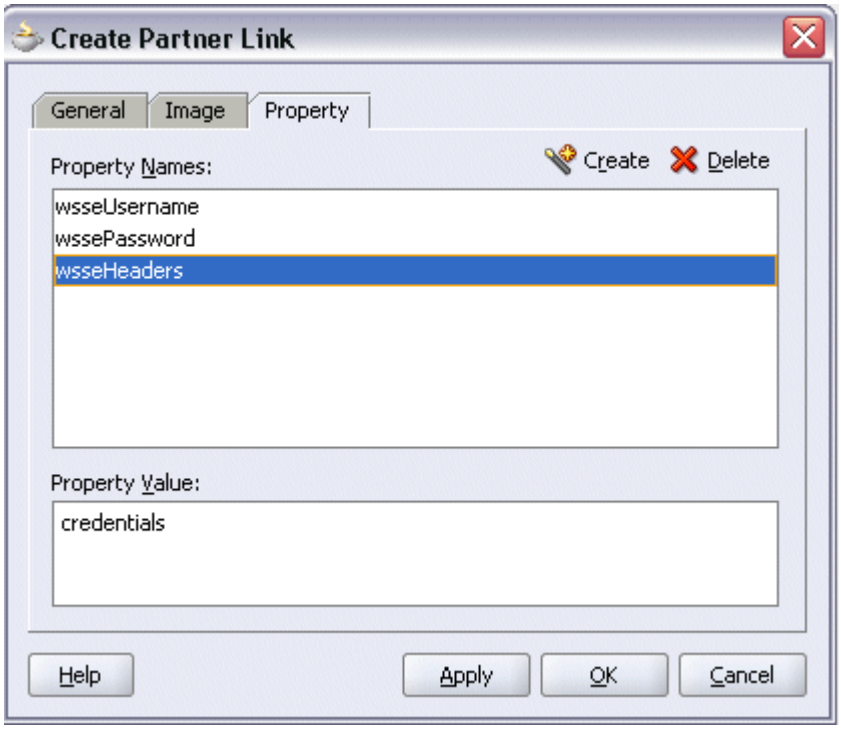
     Specify the corresponding password for the username to be passed in the Property Value box.

   - wsseHeaders

     Enter `credentials` as the property value.

   Click **Apply** to save the selected property values.

Create Partner Link: Property Tab



6. Click **OK** to complete the partner link configuration.

Adding a Partner Link for File Adapter

Use this step to configure a BPEL process to read input payload.

To add a Partner Link for File Adapter to Read Payload:

1. In JDeveloper BPEL Designer, drag and drop the **File Adapter** service from the **Adapter Service** section of the Component Palette into the Partner Link area of the process diagram. The Adapter Configuration Wizard welcome page appears.

2. Click **Next**. The Service Name dialog box appears.

3. Enter a name for the file adapter service such as `Read_Payload`. You can add an optional description of the service.

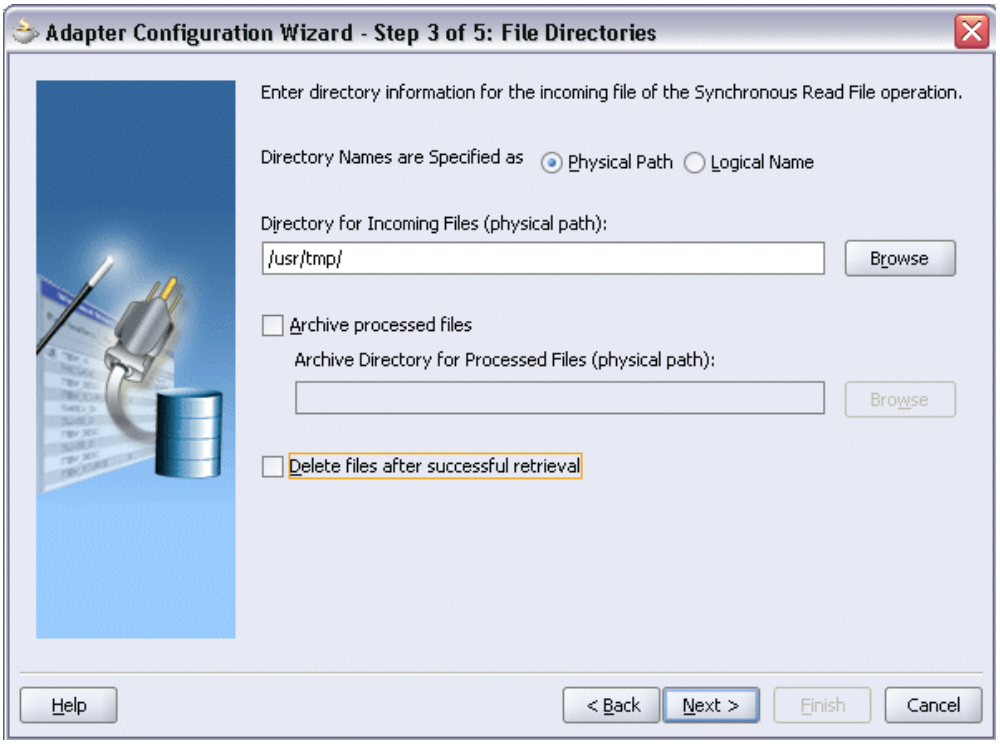
4. Click **Next**. The Operation dialog box appears.

Operation Dialog

5. Specify the operation type, for example **Synchronous Read File**. This automatically populates the **Operation Name** field.

   Click **Next** to access the File Directories dialog box.

File Directories Dialog



6. Select the **Physical Path** radio button and enter the physical path for incoming file directory information. For example, enter `/usr/tmp/`.

   > **Note:** To be able to locate the file from the physical directory you specified here, you must first place the input payload file (such as `Inputzzsdrequest.xml`) to the specified directory.

   Alternatively, click **Browse** to locate the incoming file directory information.

   Uncheck the **Delete Files after successful retrieval** check box. Click **Next** to open the File Name dialog box.

7. Enter the name of the file for the synchronous read file operation. For example, enter `Inputzzsdrequest.xml`. Click **Next**. The Messages dialog box appears.

8. Select **Browse** in the Schema Location field to open the Type Chooser dialog.

   Click **Import Schema File...** icon on the top right corner of the Type Chooser dialog.

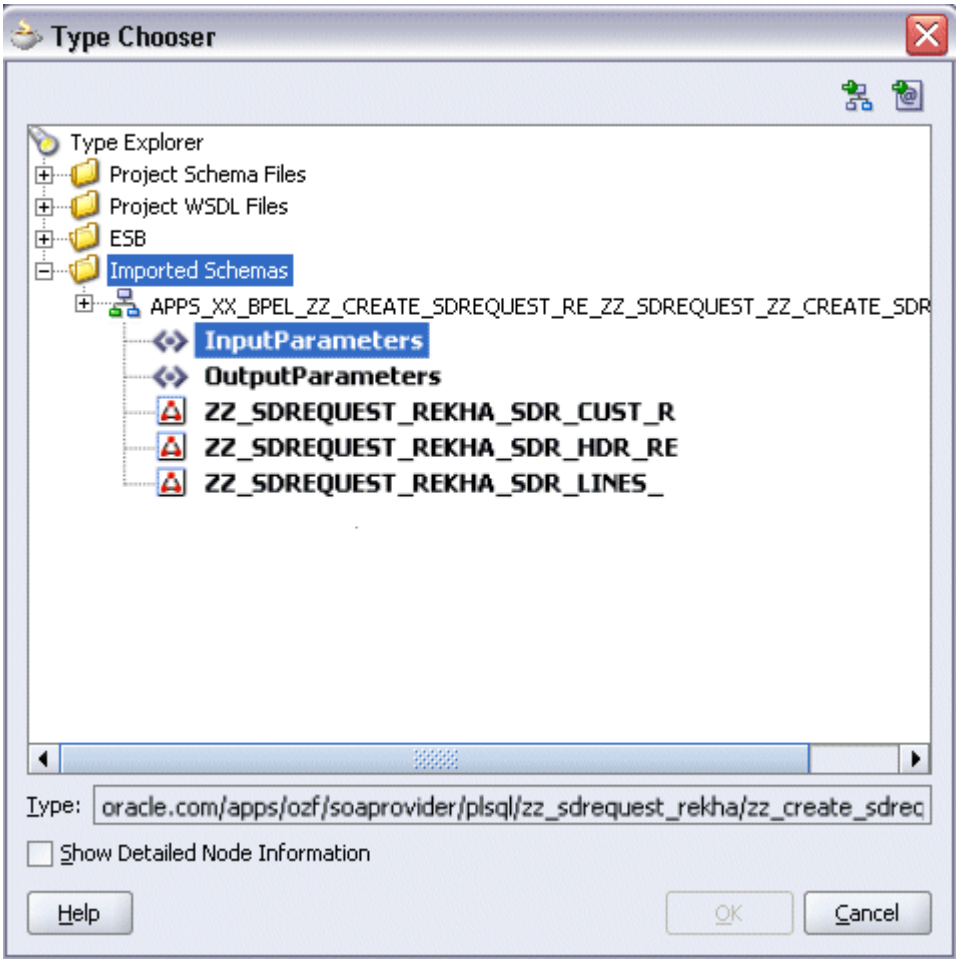
9. Enter the schema location for the custom service, such as `http://<host>:<port>/webservices/SOAProvider/plsql/zz_sdrequest/APPS_XX_BPEL_ZZ_CREATE_SDREQUEST_RE_ZZ_SDREQUEST_ZZ_CREATE_SDREQU.xsd`.

The schema location for your custom service can be found from the custom service WSDL (for example, `http://<host>:<port>/webservices/SOAProvider/plsql/zz_sdrequest/?wsdl`).

Select the 'Add to Project' check box and click **OK**.

10. Click **OK** to import schema prompt. The imported schema section will be added to the Type Choose dialog.

Type Chooser Dialog



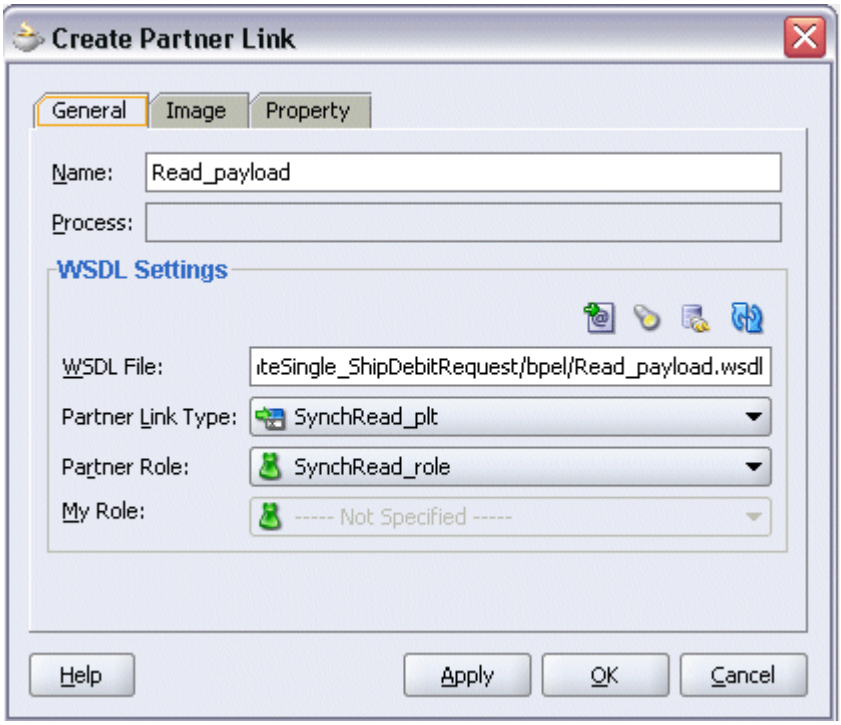
11. Browse the imported schema by selecting Imported Schemas > APPS_XX_BPEL_ZZ_CREATE_SDREQUEST_RE_ZZ_SDREQUEST_ZZ_CREATE_SDREQU.xsd > InputParameters.

    Click **OK**. The selected .xsd is displayed as Schema Location, and `InputParameters` is displayed as Schema Element.

12. Click **Next** and then **Finish**. The wizard generates the WSDL file corresponding to

the partner link. The main Create Partner Link dialog box appears, specifying the new WSDL file `Read_Payload.wsdl`.

Create Partner Link Dialog



Click **Apply** and **OK** to complete the configuration and create the partner link with the required WSDL settings for the File Adapter Service.

The `Read_Payload` Partner Link appears in the BPEL process diagram:

Adding Invoke Activities

This step is to configure two Invoke activities:

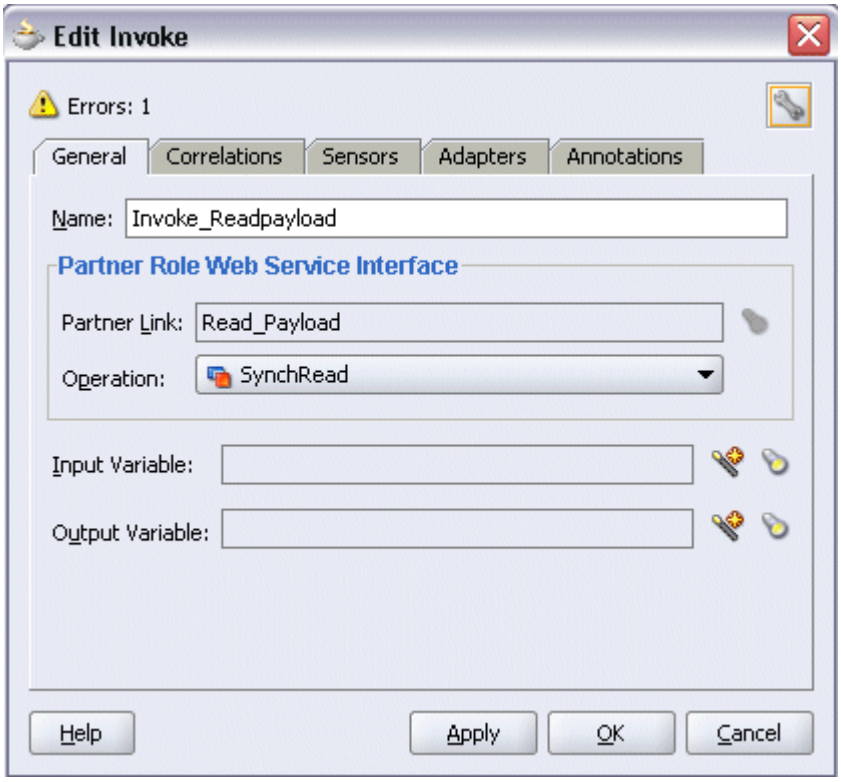
- Read supplier ship and debit request creation details that is passed from the first Assign activity using `Read_Payload` partner link for File Adapter.
- Send the payload and request number details received from the Assign activities to create a single supplier ship and debit request by using the `ZZ_SDREQUEST` partner link.

To add an Invoke activity for Read_Payload Partner Link:

1. In JDeveloper BPEL Designer, drag and drop the **Invoke** activity from the Component Palette into the Activity box of the process diagram, between the **Receive** and **Reply** activities.

2. Link the Invoke activity to the `Read_Payload` service. The Invoke activity will send request data to the partner link. The Edit Invoke dialog box appears.

Edit Invoke Dialog



3. Enter a name for the Invoke activity such as 'Invoke_Readpayload', and then click the **Create** icon next to the **Input Variable** field to create a new variable. The Create Variable dialog box appears.

4. Enter a name for the variable such as 'Invoke_Readpayload_InputVariable' and select **Global Variable**. Click **OK** in the Create Variable dialog box.

   Enter a name for the output variable such as 'Invoke_Readpayload_OutputVariable' and select **Global Variable**. Click **OK** in the Create Variable dialog box.

   Click **Apply** and **OK** in the Edit Invoke dialog box to finish configuring the Invoke activity.

   The Invoke activity appears in the process diagram.

To add an Invoke activity for ZZ_SDREQUEST Partner Link:

1. In JDeveloper BPEL Designer, drag and drop the **Invoke** activity from the Component Palette into the Activity box of the process diagram, after the **Invoke** and **Reply** activities.

2. Link the Invoke activity to the `ZZ_SDREQUEST` service. The Invoke activity will send the request number to the partner link. The Edit Invoke dialog box appears.

3. Enter a name for the Invoke activity such as 'Invoke_zzsdrequest'.

   Select the Operation as ZZ_CREATE_SDREQUEST.

4. Click the **Create** icon next to the **Input Variable** field to create a new variable such as 'Invoke_zzsdrequest_InputVariable'. Select **Global Variable** and click **OK** in the Create Variable dialog box.

5. Click the **Create** icon next to the **Output Variable** field to create a new variable such as 'Invoke_zzsdrequest_OutVariable'. Select **Global Variable** and click **OK** in the Create Variable dialog box. Click **Apply** and **OK** in the Edit Invoke dialog box to complete the Invoke activity creation.

   The Invoke activity appears in the process diagram.

Adding Assign Activities

This step is to configure four Assign activities:

1. To set the applications context information obtained from the dummy Receive activity, that will be used in passing variables for SOAHeader elements of the SOAP request.

   > **Note:** You need to populate certain variables in the BPEL process for SOAHeader elements to pass values that may be required to set applications context during service execution. These SOAHeader elements are `Responsibility, RespApplication, SecurityGroup, NLSLanguage`, and `Org_Id`.

2. To pass the payload information to the `Invoke_zzsdrequest` Invoke activity.

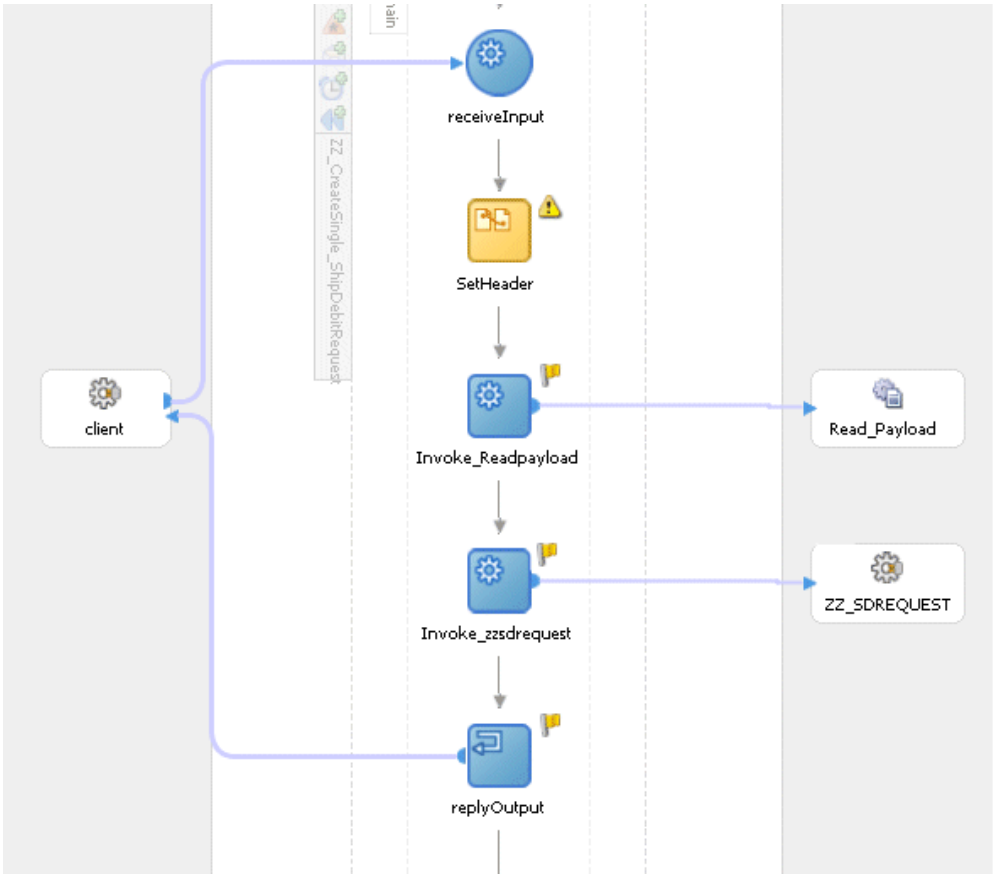
3. To pass the supplier ship and debit request number information to the `Invoke_zzsdrequest` Invoke activity.

4. To pass the supplier ship and debit request number information back to the dummy Reply activity as an output.

To add the first Assign activity to pass applications context details to the Invoke_Readpayload Invoke activity:

1. In JDeveloper BPEL Designer, drag and drop the **Assign** activity from the Component Palette into the Activity box of the process diagram, between the **Receive** activity and the first **Invoke** activity.

BPEL Process Dialog with Assign Activity



2. Double-click the **Assign** activity to access the Edit Assign dialog box.

3. Click the General tab to enter the name for the Assign activity, such as 'SetHeader'.

4. On the Copy Operation tab, click **Create** and then select **Copy Operation** from the menu. The Create Copy Operation window appears.

5. Enter the first pair of parameters:

   - In the From navigation tree, select type Expression and then enter '204' in the Expression box.

   - In the To navigation tree, select type Variable. Navigate to **Variable > Process > Variables > Invoke_zzsdrequest_InputVariable >header> ns5:SOAHeader** and select **ns5:ORG_ID**. The XPath field should contain your selected entry.

   - Click **OK**.

6. Enter the second pair of parameters by selecting **Copy Operation** from the Create drop-down list with the following values:

    - In the From navigation tree, select type Expression and then enter 'TRADE_MANAGEMENT_USER' in the Expression box.

    - In the To navigation tree, select type Variable. Navigate to **Variable > Process > Variables > Invoke_zzsdrequest_InputVariable >header > ns5:SOAHeader** and select **ns5:Responsibility**. The XPath field should contain your selected entry.

    - Click **OK**.

7. Enter the third pair of parameters by selecting **Copy Operation** from the Create drop-down list with the following values:

    - In the From navigation tree, select type Expression and then enter 'OZF' in the Expression box.

    - In the To navigation tree, select type Variable. Navigate to **Variable > Process > Variables > Invoke_zzsdrequest_InputVariable >header > ns5:SOAHeader** and select **ns5:RespApplication**. The XPath field should contain your selected entry.

    - Click **OK**.

8. Enter the fourth pair of parameters by selecting **Copy Operation** from the Create drop-down list with the following values:

    - In the From navigation tree, select type Expression and then enter 'STANDARD' in the Expression box.

    - In the To navigation tree, select type Variable. Navigate to **Variable > Process > Variables > Invoke_zzsdrequest_InputVariable >header > ns5:SOAHeader** and select **ns5:SecurityGroup**. The XPath field should contain your selected entry.

    - Click **OK**.

9. Enter the fifth pair of parameters by selecting **Copy Operation** from the Create drop-down list with the following values:

    - In the From navigation tree, select type Expression and then enter 'AMERICAN' in the Expression box.

    - In the To navigation tree, select type Variable. Navigate to **Variable > Process > Variables > Invoke_zzsdrequest_InputVariable >header > ns5:SOAHeader**

and select **ns5:NLSLanguage**. The XPath field should contain your selected entry.

- Click **OK**.

10. The Edit Assign dialog box appears.

11. Click **OK** to complete the configuration of the Assign activity.

To enter the second Assign activity to pass payload information to the Invoke_zzsdrequest Invoke activity:

1. Add the second Assign activity by dragging and dropping the **Assign** activity from the Component Palette into the Activity box of the process diagram, between two **Invoke** activities.

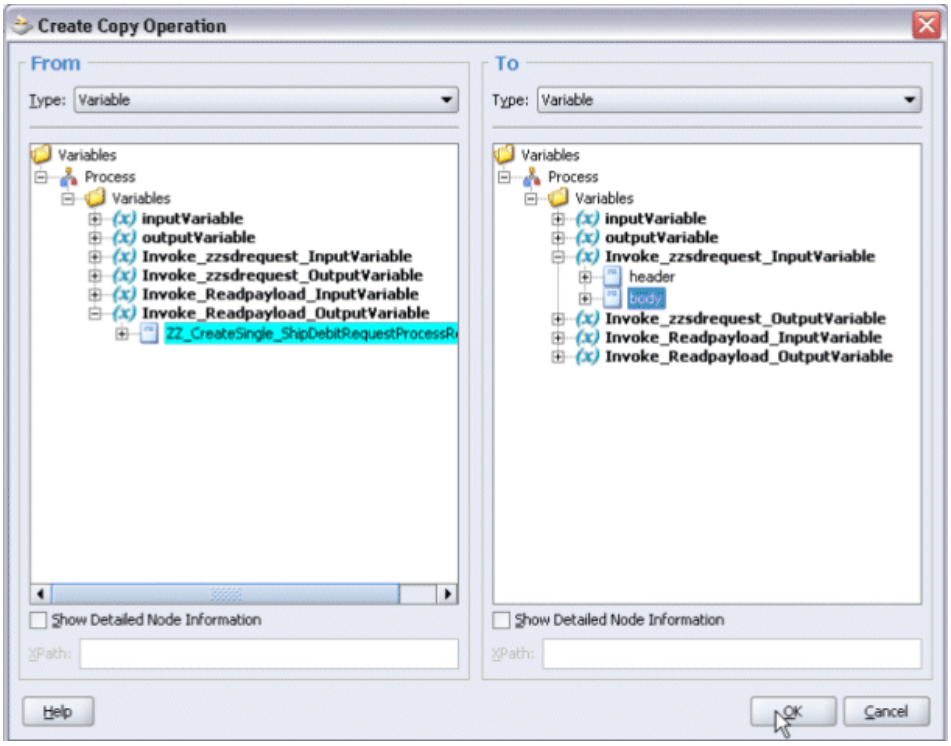
2. Repeat Step 2 to Step 4 described in creating the first Assign activity to add the second Assign activity called 'SetPayload'.

3. Enter the following information:

   - In the From navigation tree, navigate to **Variable > Process > Variables > Invoke_ReadPayload_OutVariable > ZZ_CreateSingle_ShipDebitRequestProcessRequest**.

   - In the To navigation tree, select type Variable. Navigate to **Variable > Process > Variables > Invoke_zzsdrequest_InputVariable > Body**.

Create Copy Operation Dialog



Click **OK** in the Create Copy Operation window.

4. Click **OK** to complete the configuration of the Assign activity.

To enter the third Assign activity to pass the supplier ship and debit request number to the Invoke_zzsdrequest Invoke activity:

1. Add the third Assign activity by dragging and dropping the **Assign** activity from the Component Palette into the Activity box of the process diagram, between the second **Assign** activity and the `Invoke_zzsdrequest` **Invoke** activity.

2. Repeat Step 2 to Step 4 described in creating the first Assign activity to add the third Assign activity called 'SetRequestNumber'.

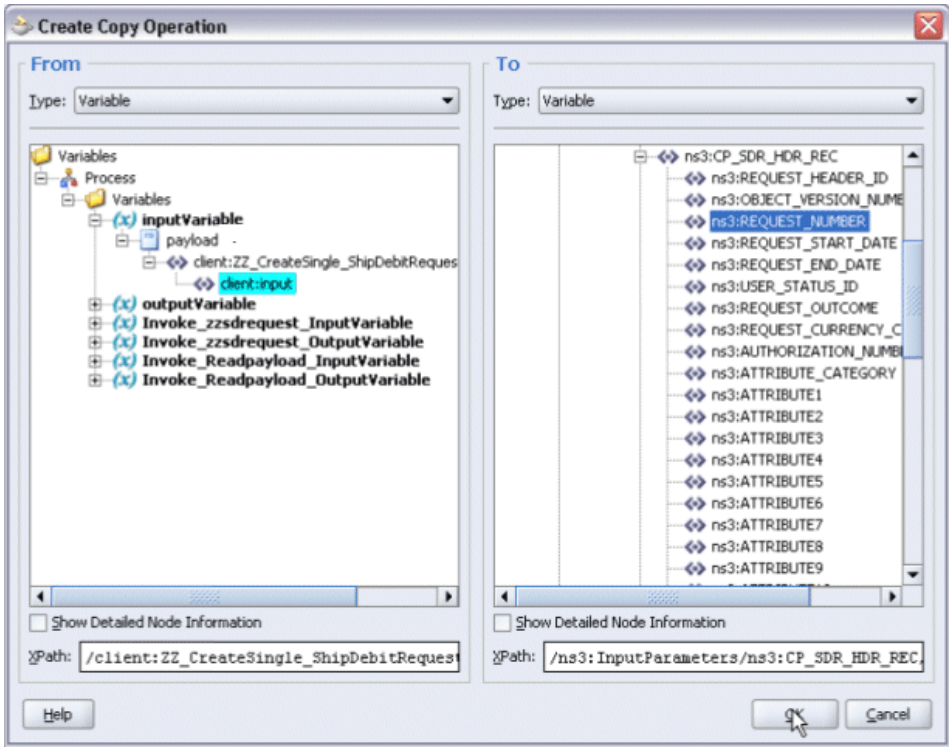
3. Enter the following information:

   - In the From navigation tree, navigate to **Variable > Process > Variables > inputVariable > Payload > client: ZZ_CreateSingle_ShipDebitRequestProcessRequest > client:input**. The XPath field should contain your selected entry.

   - In the To navigation tree, select type Variable. Navigate to **Variable > Process >**

Variables > Invoke_zzsdrequest_InputVariable > Body > ns3:InputParameters >ns3:CP_SDR_HDR_REC > ns3:REQUEST_NUMBER and select **ns3: TRX_NUMBER**. The XPath field should contain your selected entry.

Create Copy Operation Dialog



- Click **OK** in the Create Copy Operation window.

4. Click **OK** in the Assign window to complete the configuration of the Assign activity.

To add the fourth Assign activity to reply back supplier ship and debit request number:

1. Add the third Assign activity by dragging and dropping the **Assign** activity from the Component Palette into the Activity box of the process diagram, between the `Invoke_zzsdrequest` **Invoke** and the **Reply** activities.

2. Repeat Step 2 to Step 4 described in creating the first Assign activity to add the fourth Assign activity called 'SetRequestNumber'.

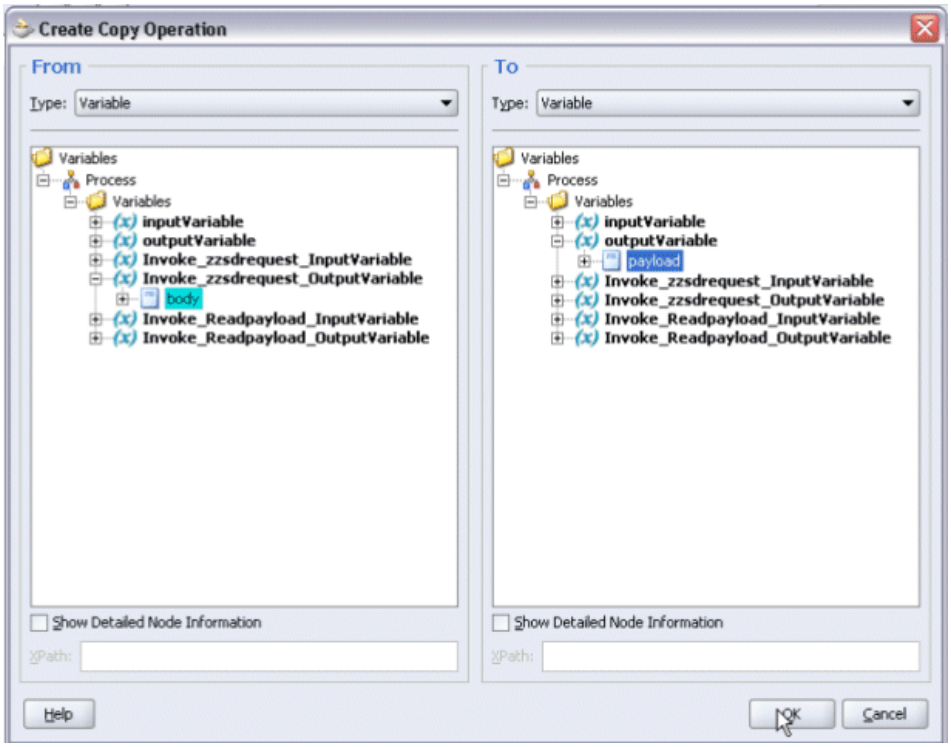
3. Enter the following information:

   - In the From navigation tree, select type Variable. Navigate to **Variable >**

Process > Variables > Invoke_zzsdrequest_OutputVariable > Body.

- In the To navigation tree, select type Variable. Navigate to **Variable > Process > Variables > outputVariable > payload**.

Create Copy Operation Dialog



4. Click **OK** in the Create Copy Operation window.

5. Click **OK** in the Assign window to complete the configuration of the Assign activity.

Deploying and Testing the BPEL Process at Run Time

After creating a BPEL process using the WSDL URL generated from a custom PL/SQL interface definition, you can deploy it to a BPEL server if needed. To ensure that this process is modified or orchestrated appropriately, you can also manually test the BPEL process by initiating the business process contained in the BPEL process to test the interface integration.

Prerequisites

Before deploying the BPEL process using Oracle JDeveloper, you must ensure that you have established the connectivity between the deign-time environment and the run-time servers including the application server and the integration server.

How to configure the necessary server connection, see Configuring Server Connection, page B-1.

To validate your BPEL process, perform the following run-time tasks:

1. Deploy the BPEL process, page 11-44

   Once you deploy the process to a BPEL server, it becomes available so that you can run the process manually to test it for validation.

2. Test the BPEL process, page 11-45

   After deploying a BPEL process, you can manage the process from the BPEL console to manually initiate the business process and test the interface integration contained in your BPEL process.

Deploying the BPEL Process

You must deploy the Create Single Supplier Ship and Debit Request BPEL process (`ZZ_CreateSingle_ShipDebitRequest.bpel`) that you created earlier before you can run it.

To deploy the BPEL process:

1. In the Applications Navigator of JDeveloper BPEL Designer, select the **ZZ_CreateSingle_ShipDebitRequest** project.

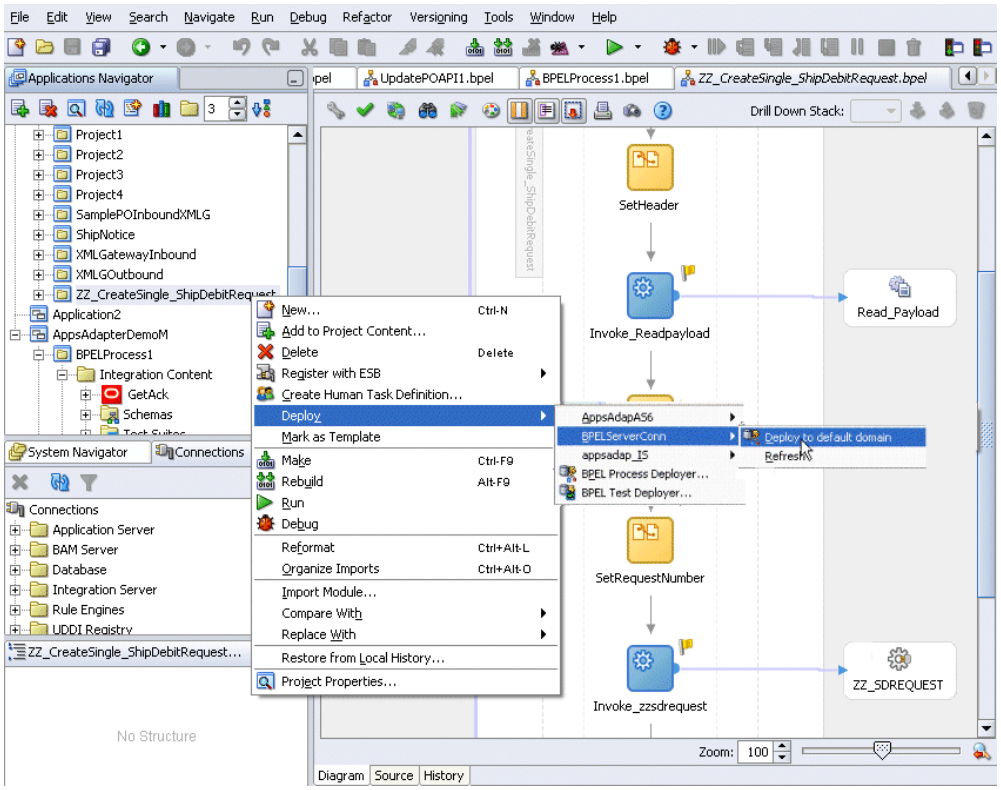
2. Right-click the project and select **Make** action from the menu to ensure the successful server connections.

   You can look for any compilation error messages in Messages Log.

3. Right-click the project and select **Deploy** action from the menu. Click on **IntegrationServerConnection name** and enter your BPEL Process Manager information.

   For example, you can select **Deploy > BPELServerConn > Deploy to Default Domain** to deploy the process if you have the BPEL Process Manager set up appropriately.

Oracle JDeveloper for Deploying the BPEL Process



4. The Password Prompt dialog box appears.

   Enter the password for the default domain in the **Domain Password** field and click **OK**.

   The BPEL project is compiled and successfully deployed.

Testing the BPEL Process

To validate whether the BPEL process that you created works or not, you need to manually initiate the process after it has been successfully deployed to the BPEL server. Therefore, the validation starts with the BPEL console to ensure that you can find the deployed BPEL process listed in the console. Then, you can log on to Oracle E-Business Suite to validate that the supplier ship and debit request is successfully created with the request number you specified.

To test the BPEL process:

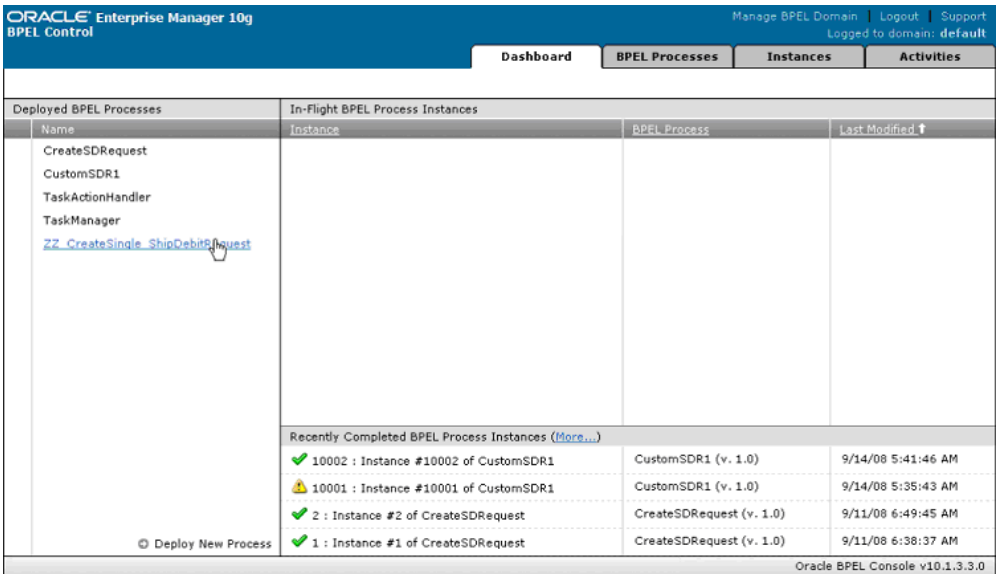
1. Log in to Oracle Application Server 10g BPEL Console (`http://<soaSuiteServerHostName>:<port>/BPELConsole`). The BPEL Console login page appears.

2. Enter the username and password and click **Login**.

   The Oracle Enterprise Manager 10g BPEL Control appears.

3. In the BPEL Console, confirm that ZZ_CreateSingle_ShipDebitRequest has been deployed.

Oracle Enterprise Manager BPEL Control Console with Deployed BPEL Processes

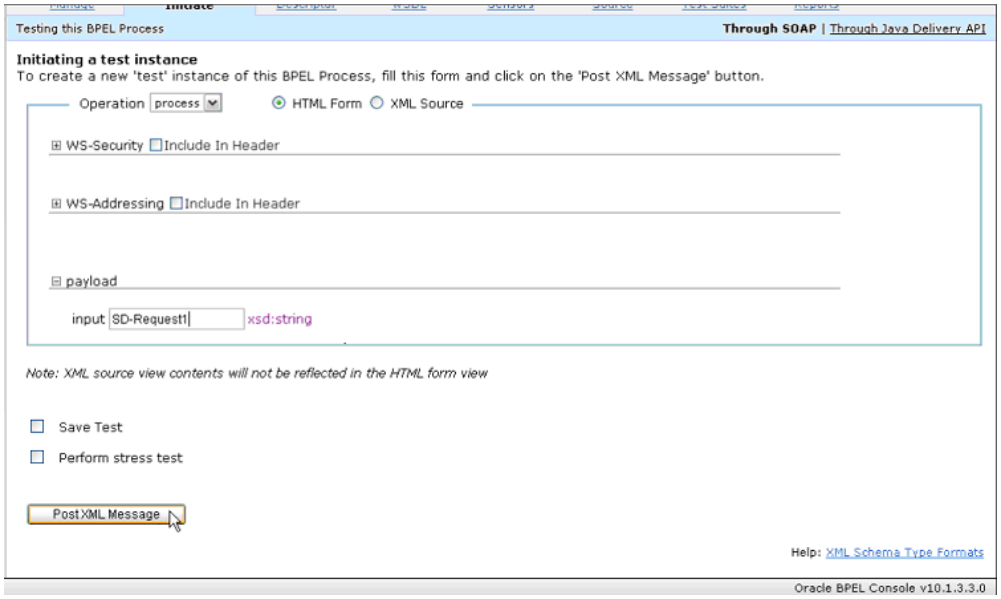


4. Click the ZZ_CreateSingle_ShipDebitRequest link to open the Initiate tab.

5. Enter Payload input field, such as 'SD-Request1' and click **Post XML Message** to initiate the process.

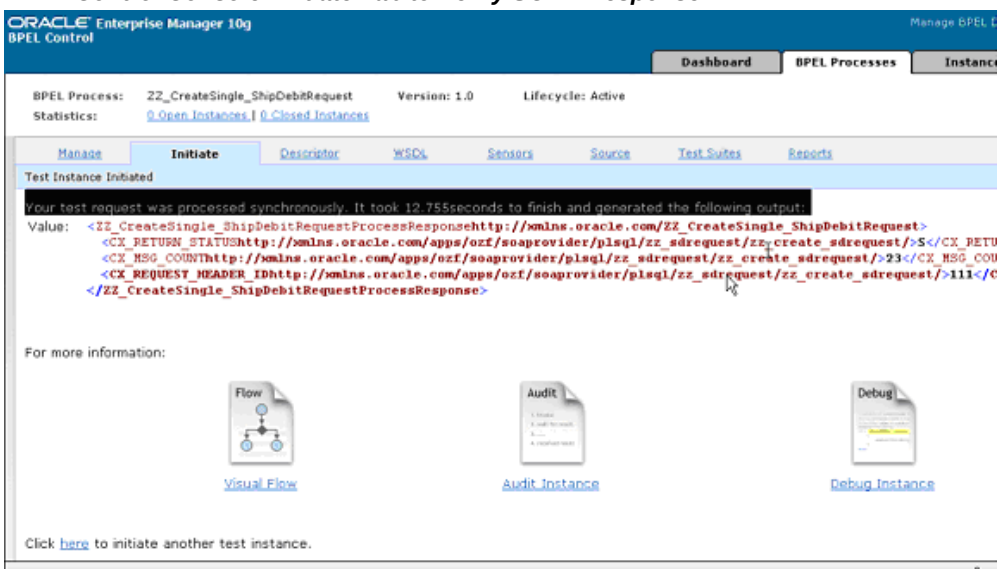
   Note: The Request Number entered here should be unique each time that you initiate the process because this number will be used as the Supplier Ship and Debit number across users in Oracle Trade Management.

BPEL Control Console: Initiate Tab to Enter Payload Information



6. You can verify SOAP Response in BPEL Console.

BPEL Control Console: Initiate Tab to Verify SOAP Response



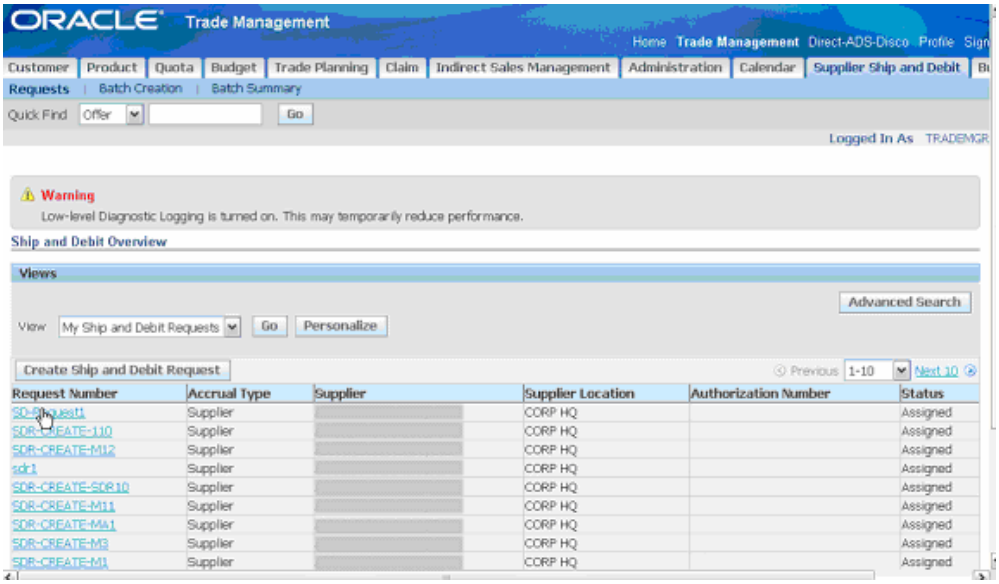
Look for 'S' in CX_RETURN_STATUS for success. If 'E' is displayed instead, then it means error has occurred while processing the service. Look for detailed exception message in SOA Monitor.

7. Log in to Oracle E-Business Suite as a user who has the Oracle Trade Management

User responsibility. Select the Oracle Trade Management User responsibility from the navigator. Select the 'Supplier Ship and Debit' link to open the Ship and Debit Overview window.

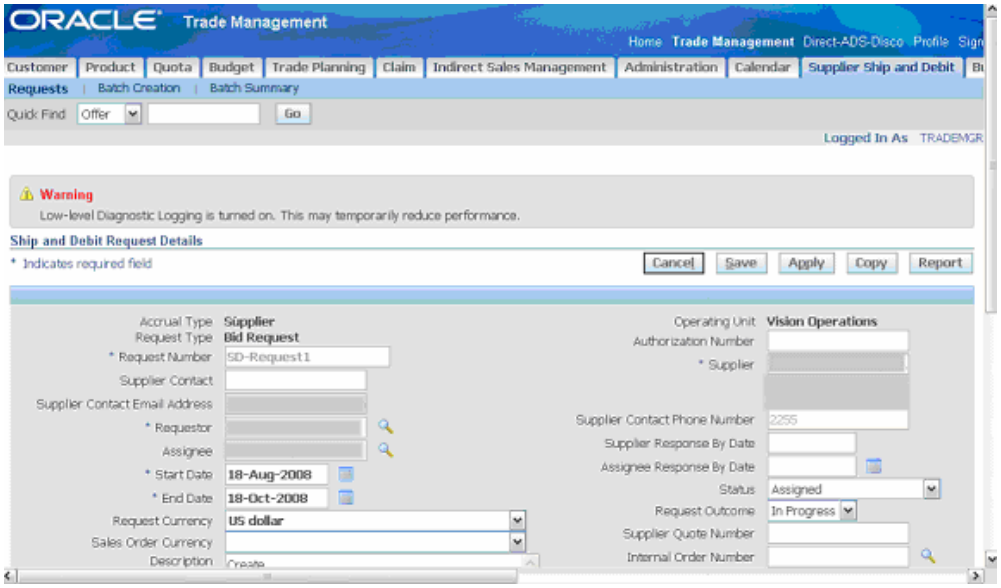
8. Verify if the request number 'SD-Request1' that you entered in Step 5 appears in the list.

Ship and Debit Overview Window



9. Click the request number 'SD-Request' link to open the Ship and Debit Request Details window. You can verify the request details.

Ship and Debit Request Details Window



12

Working With Oracle Workflow Business Event System to Invoke Web Services

This chapter covers the following topics:

- Oracle Workflow and Service Invocation Framework Overview
- Web Service Invocation Using Service Invocation Framework
- Calling Back to Oracle E-Business Suite With Web Service Response
- Invoking Web Services
- Managing Errors
- Testing Web Service Invocation
- Troubleshooting Web Service Invocation Failure
- Extending Seeded Java Rule Function
- Other Invocation Usage Considerations

Oracle Workflow and Service Invocation Framework Overview

Oracle E-Business Suite Integrated SOA Gateway leverages Oracle Workflow Java Business Event System to provide infrastructure for Web Service Invocation natively from Oracle E-Business Suite.

Oracle Workflow is the primary process management solution within Oracle E-Business Suite. It consists of some key components enabling you model and automate business processes and activities in a process diagram based on user-defined business rules, providing routing mechanism to support each decision maker in the process, facilitating subscriptions to significant events or services between systems, and implementing workflow process definitions at run time with monitoring capability of each workflow state as well as handling errors. Since it provides a total solution of managing and streamlining complex business processes and supporting highly-integrated workflow in and out from Oracle E-Business Suite, Oracle E-Business Suite Integrated SOA Gateway relies on Oracle Workflow to enable the service invocation process and provide the

following functionality:

- It relies on Business Event System to create events and event subscriptions as well as to parse a given WSDL representing a Web service to be consumed as subscription parameters.
- It uses the Oracle Workflow seeded Java rule function `oracle.apps.fnd.wf.bes.WebServiceInvokerSubscription` to help invoke Web services.
- It relies on the Oracle Workflow Test Business Event page to test service invocation by raising an invoker event raised from PL/SQL or Java and execute synchronous and asynchronous subscriptions to the event.
- It utilizes the Error processing feature provided in Business Event System to manage errors during subscription execution and sends error notifications to `SYSADMIN` user with Web service definition, error and event details.
- It utilizes workflow Notification System to send error notifications to and process responses from `SYSADMIN`.

For detailed information about Oracle Workflow, see *Oracle Workflow User's Guide*, *Oracle Workflow Developer's Guide*, and *Oracle Workflow Administrator's Guide*.

To better understand how service invocation framework is used in facilitating the invocation of Web services, the following topics are discussed in this chapter:

Web Service Invocation Using Service Invocation Framework

Service invocation framework provides an infrastructure allowing developers to interact with Web services through WSDL descriptions and to invoke Web services from Oracle E-Business Suite.

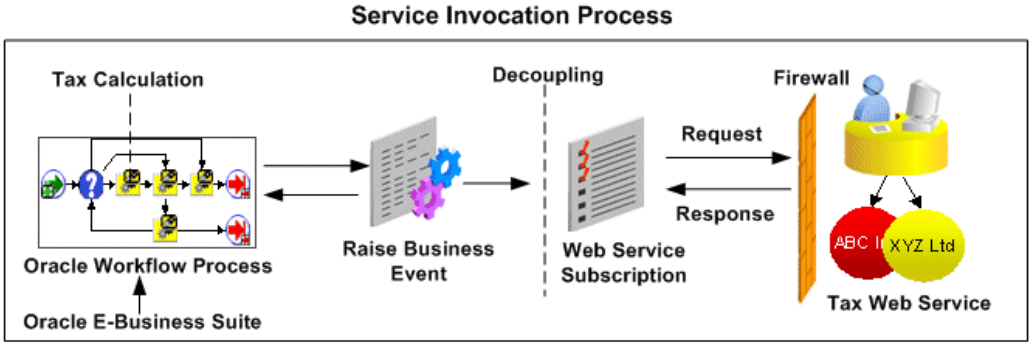
To achieve this goal, the invocation framework uses a wizard based user interface in

Oracle Workflow Business Event System to parse a given Web service WSDL URL during the subscription creation and store identified service information or metadata as subscription parameters that will be used later during service invocation.

Since a WSDL URL is used in representing a Web service, the underlying service can be a simple native Web Service or it can be a BPEL process.

Please note that the service invocation framework discussed here only supports document-based Web service invocation. The invocation framework does not support RPC (remote procedure call) style Web service invocation.

The following diagram illustrates the high level service invocation process flow:



To successfully invoke Web services at run time, Web service invocation metadata must first be in place. In addition to defining the invocation metadata, the concepts of message patterns, Web service input message parts, and Web service security that the service invocation framework supports are also introduced in this section.

The section covers the following topics:

- Understanding Message Patterns, page 12-3
- Defining Web Service Invocation Metadata, page 12-5
    1. Creating a Web Service Invoker Business Event, page 12-6
    2. Creating Local and Error Event Subscriptions to the Invoker Event, page 12-8
    3. Creating a Receive Event and Event Subscription (Optional), page 12-18
- Understanding Web Service Input Message Parts, page 12-22
- Supporting WS-Security, page 12-27

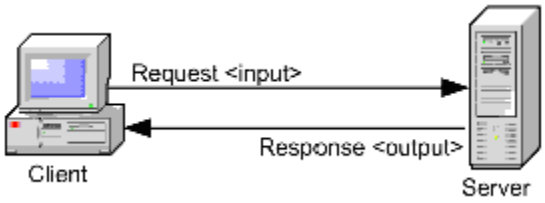
Understanding Message Patterns in WSDL

There are two major message exchange patterns — a request-response pattern, and a one-way (request - only) pattern.

Request - Response Message Pattern

The *request - response* message exchange pattern is where a client asks a service provider a question and then receives the answer to the question. The answer may come in the form of a fault or exception. Both the request and the response are independent messages. The request - response pattern is often implemented using synchronous operations for simple operations. For longer running operations, asynchronous (with message correlation) is often chosen.

Request - Response Message Pattern



- A *synchronous* operation is one that waits for a response before continuing on. This forces operations to occur in a serial order. It is often said that an operation, "blocks" or waits for a response. Many online banking tasks are programmed in request/response mode.

  For example, a request for an account balance is executed as follows:

  - A customer (the client) sends a request for an account balance to the Account Record Storage System (the server).
  - The Account Record Storage System (the server) sends a reply to the customer (the client), specifying the dollar amount in the designated account.

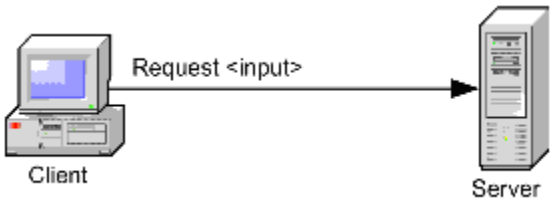
- An *asynchronous* operation is one that does not wait for a response before continuing on. This allows operations to occur in parallel. Thus, the operation does not, "block" or wait for the response. Asynchronous operations let clients continue to perform their work while waiting for responses that may be delayed. This is accomplished by returning an asynchronous handle that runs a thread in the background, allowing the client to continue execution until the response is ready.

  Important: In this release, the Web service invocation framework only supports Synchronous Request - Response message pattern and One - Way (Request Only) message pattern.

Request Only Message Pattern

The *request only* operation model includes one **input** element, which is the client's request to the server. No response is expected.

Request Only Message Pattern



For example, client zip code locations send updated weather data to the service when local conditions change using the request only operation. The server updates the data but no response is sent back.

Defining Web Service Invocation Metadata

Because the service invocation is taken place in the Oracle Workflow Business Event System, before invoking a Web service, the Web service invocation metadata including events and event subscriptions must be defined first through the Business Event System.

This section discusses the following topics:

1. **Creating a Web Service Invoker Business Event**, page 12-6

   A Web service Invoker business event that serves as a request message (or Web service input message) for a service needs to be created first.

2. **Creating Local and Error Event Subscriptions to the Invoker Event**, page 12-8

   After defining the Invoker event, you need to create the following two subscriptions:

   - Create a Local subscription with 'Invoke Web Service' Action Type, page 12-8

     This event subscription indicates that when a triggering event occurs, the action item of this subscription is to invoke a Web service defined as part of this subscription.

   - Create an Error subscription with 'Launch Workflow' Action Type, page 12-15

     This error subscription enables error processing in the Business Event System that is used to communicate with SYSADMIN user of an error condition in subscription execution.

3. **Creating a Receive Event and Event Subscription (Optional)**, page 12-18

   This step is required only if a Web service has an output or a response message to communicate or callback to Oracle E-Business Suite. Once a receive event is in place, you must create an External subscription to the receive event to pass the Web

service response message.

If a Web service does not require a response, then you do not need to create a receive event, nor the event subscription.

Step 1: Creating a Web Service Invoker Business Event

A business event is an occurrence in an internet or intranet application or program that might be significant to other objects in a system or to external agents. For instance, the creation of a purchase order is an example of a business event in a purchasing application.

Use the Oracle Workflow Business Event System to define a web service invoker business event.

The invoker event can be served as a request message (or web service input message) in a message pattern to send inquiries to a service.

To invoke a web service through the Business Event System, we will first create an invoker business event, and then subscribe to the invoker event later with an appropriate action type.

> **Note:** In this release, the web service invocation framework supports the following types of service invocation:
>
> - **One-way (request only)** service that a consumer or client sends a message to a service, and the service does not need to reply.
>
> - **Synchronous request-response** service type that requires a response before an operation continues.
>
>   If an invoker event requires a response, then you must define a receive business event to communicate or callback into Oracle E-Business Suite after the web service is successfully invoked. See Creating a Receive Event and Event Subscription (Optional), page 12-18.

For more information about business events, see Events, *Oracle Workflow Developer's Guide*.

To create an invoker event:

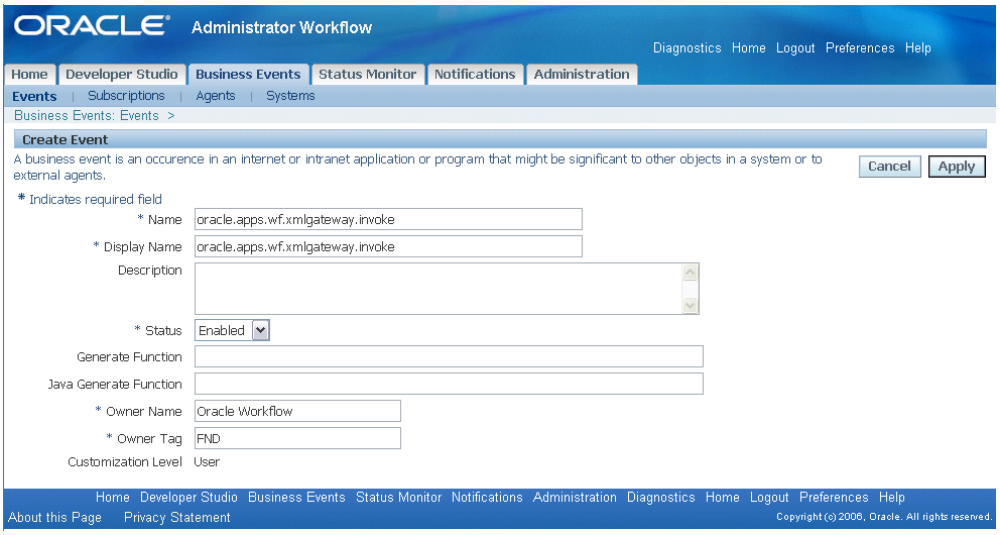
1. Log on to Oracle E-Business Suite with the Workflow Administrator Web responsibility. Select the Business Events link, and choose Events in the horizontal navigation if the Events page is not already displayed.

2. In the Events page, click **Create Event** to open the Create Event page.

3. Enter the following information in the Create Event page:

- Name: Enter an event name, such as `oracle.apps.wf.xmlgateway.invoke`
- Display Name: Enter an event display name, such as `oracle.apps.wf.xmlgateway.invoke`
- Description: Enter a description for the event
- Status: Enabled
- Generate Function: Specify a generate function for the PL/SQL based event if the application where the event occurs will not provide the event data
- Java Generate Function: Specify a generate function for the Java based event if the application where the event occurs will not provide the event data
- Owner Name: Specify the program or application name that owns the event (such as Oracle Workflow)
- Owner Tag: Specify the program or application ID that owns the event (such as 'FND')

Create Event Page



4. Click **Apply** to save your work.

   Leave this page open to create a receive event.

For more information on how to create a business event, see *Oracle Workflow Developer's Guide* for details.

Step 2: Creating Local and Error Event Subscriptions to the Invoker Event

- **Create a Local subscription with 'Invoke Web Service' Action Type**, page 12-8

  This event subscription indicates that when a triggering event occurs, the action item of this subscription is to invoke a web service that you have created in the invoke event.

- **Create an Error subscription with 'Launch Workflow' Action Type**, page 12-15

  This error subscription enables error processing in the Business Event System that is used to communicate with SYSADMIN user of an error condition in subscription execution.

  It sends a workflow notification to SYSADMIN with web service definition, error details, and event details allowing the SYSADMIN to process the errors if needed.

Create a Local Subscription With 'Invoke Web Service' Action Type

To subscribe to an invoker event, you must create a subscription with 'Invoke Web Service' Action Type which indicates that when a triggering event occurs, the action item of this subscription is to invoke a web service. This requires you enter a WSDL URL representing a web service of any type (such as a native web service or BPEL process) in the Create Event Subscription - Invoke Web Service wizard. That WSDL information entered in the wizard will then be parsed into service metadata for further selections.

> **Note:** A BPEL process itself is a web service, defining and supporting a client interface through WSDL and SOAP. The BPEL process WSDL URL can be created through a partner link which allows the request to be published to the Oracle BPEL Process Manager to connect to web services.
>
> When a triggering event occurs, the Business Event System executes the subscription through the seeded Java function and invokes the BPEL process.

After you select appropriate service metadata, this selected data will be stored as subscription parameters as follows:

- SERVICE_WSDL_URL
- SERVICE_NAME
- SERVICE_PORT
- SERVICE_PORTYPE
- SERVICE_OPERATION

The seeded Java Rule Function `oracle.apps.fnd.wf.bes.WebServiceInvokerSubscription` uses these subscription parameters during the service invocation.

> **Note:** Oracle E-Business Suite Integrated SOA Gateway allows developers to extend the invoker subscription seeded rule function using Java coding standards for more specialized service invocation processing. For more information on customizing seeded Java rule function, see Extending Seeded Rule Function, page 12-49.

Apart from the subscription parameters that have been parsed and stored through the Invoke Web Service Subscription page, the following information could also be captured if it is specified as additional subscription parameters that will then be used by the seeded Java rule function to enable message processing for web service invocation:

- Message transformation

  If the invoker event's XML payload (to be used as web service input message) requires to be transformed into a form that complies with the input message schema, the seeded Java rule function could perform XSL transformation on the payload before invoking the web service. Similarly, if the web service output message requires to be transformed into a form that is required for processing by Oracle E-Business Suite, the seeded Java rule function could perform XSL transformation on the response before calling back to Oracle E-Business Suite.

  - WFBES_OUT_XSL_FILENAME
  - WFBES_IN_XSL_FILENAME

  After event payload is either passed during the event raise or generated by generate function after the event raise, the seeded Java rule function uses these subscription parameters to obtain the XSL file names if XSL transformations are required on the web service input and output messages. At runtime, if event parameters are passed with the same names, then the event parameters override the subscription parameters.

  For more information on these transformation parameters, see Understanding Web Service Input Message Parts, page 12-22.

- WS-Security: Information required to add UsernameToken header to a SOAP request.

  If the web service being invoked enforces Username/Password based authentication, then the service invocation framework also supports the UsernameToken based WS-Security header during web service invocation. The SOAP username and optional password locator information will be passed to the seeded Java rule function as the following subscription parameters when the Java rule function is defined through the Invoke Web Service wizard:

- WFBES_SOAP_USERNAME
- WFBES_SOAP_PASSWORD_MOD
- WFBES_SOAP_PASSWORD_KEY

For more information on these WS-Security parameters, see Supporting WS-Secuirty, page 12-27.

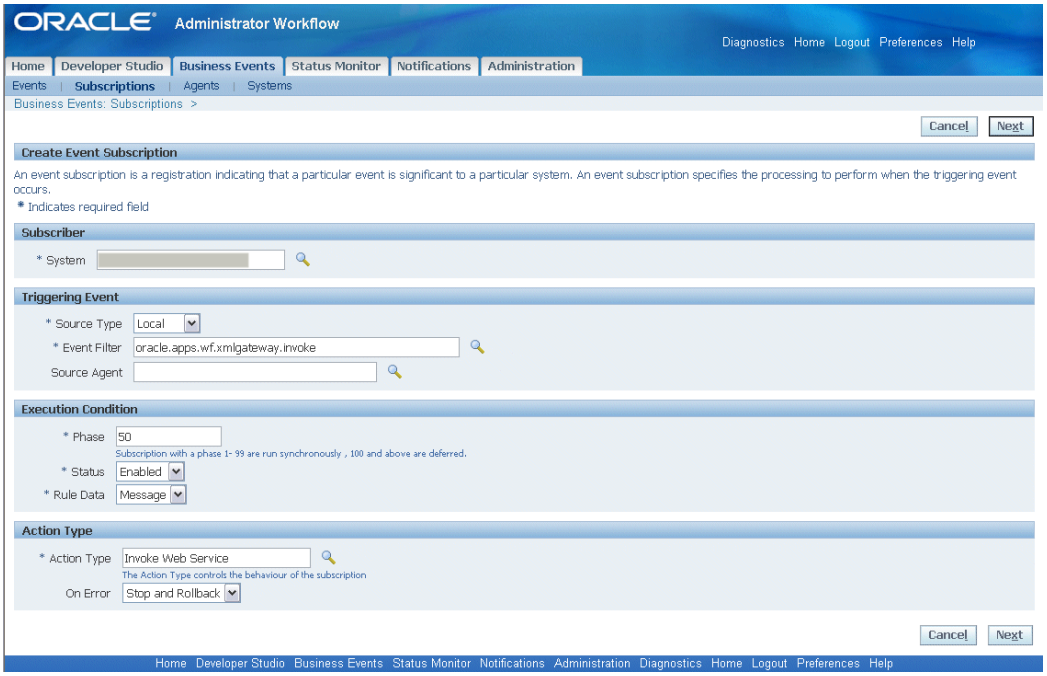
- Callback: Callback to Oracle E-Business Suite with web service response
  - WFBES_CALLBACK_EVENT
  - WFBES_CALLBACK_AGENT

  To process a web service output or response (synchronous request - response) message, the callback mechanism is used to communicate the response using a business event back to Oracle E-Business Suite by enqueueing the event to an Inbound Workflow Agent. A new or waiting workflow process can be started or executed.

  For more information on these callback parameters, see Calling Back to Oracle E-Business Suite With Web Service Response, page 12-29.

Creating a Local Event Subscription with 'Invoke Web Service' Action Type

Create Event Subscription Page



To create a local event subscription with 'Invoke Web Service' action type:

1. Log on to Oracle E-Business Suite with the Workflow Administrator Web responsibility. Select the Business Events link, and choose Subscriptions in the horizontal navigation.

2. In the Event Subscriptions page, click **Create Subscription** to open the Create Event Subscription page.

3. Enter the following information in the Create Event Subscription page:

   - Subscriber: Select the local system

   - Source Type: Local

   - Event Filter: Select the event name that you just created, such as `oracle.apps.wf.xmlgateway.invoke`

   - Phase: 50

     If the event is raised from Java, the phase number determines whether an event will be invoked right away or enqueued to WF_JAVA_DEFFERED queue.

     Note: If the invoker event is raised from PL/SQL, it is always deferred to WF_JAVA_DEFERRED queue regardless of the

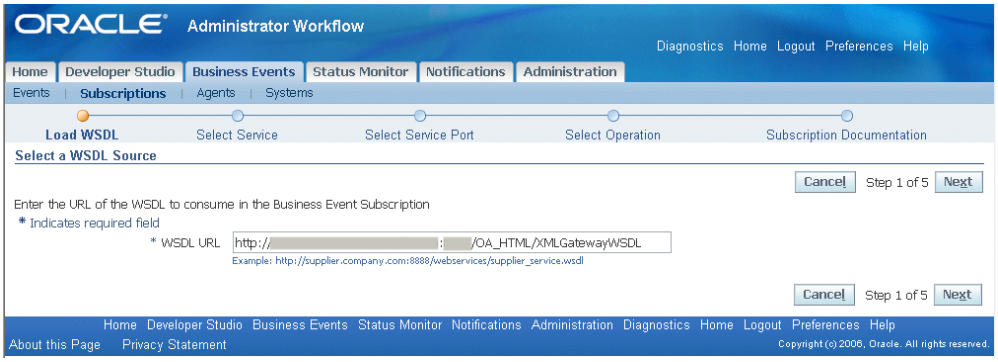
phase because the subscription has a Java rule function that cannot be executed in the database.

- If the phase is >=100, then the event is enqueued to WF_JAVA_DEFFERED queue and will be dispatched later.
- If the phase is <100, then the event is dispatched immediately to the Java Business Event System soon after an triggering event occurs.

- Status: Enabled
- Rule Data: Message
- Action Type: Invoke Web Service
- On Error: Stop and Rollback

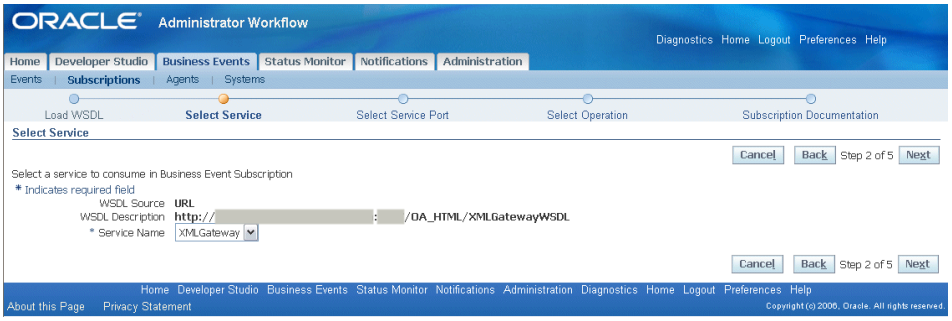
4. Click **Next**. This opens a Create Event Subscription - Invoke Web Service wizard allowing you to enter a WSDL URL that will be parsed into service metadata for further selection.

Create Event Subscription - Invoke Web Service Wizard



1. Enter WSDL URL information for the web service to be invoked. Click **Next** to parse the WSDL and display all services.
2. Select an appropriate service name from the drop-down list.

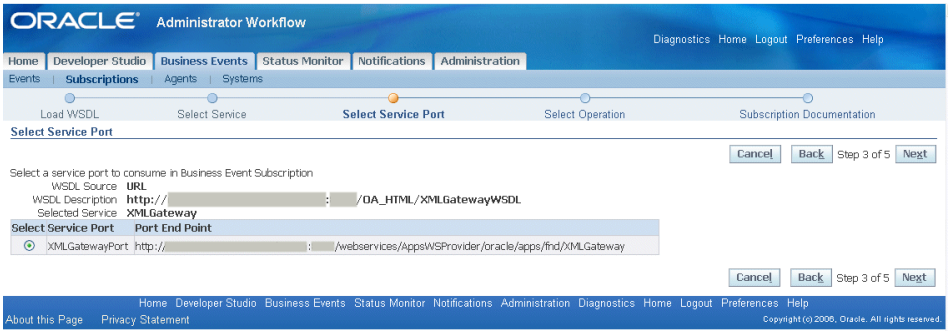
Create Event Subscription - Invoke Web Service: Select Service



Click **Next** to display all ports for a selected service.

3. Select an appropriate service port.

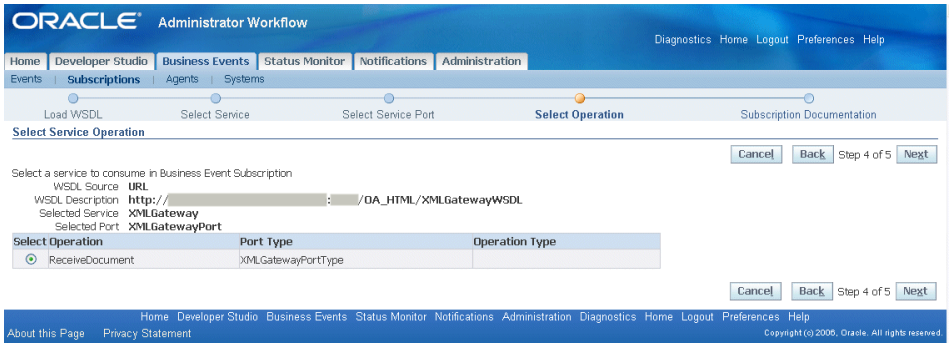
Create Event Subscription - Invoke Web Service: Select Service Port



Click **Next** to display all operations for a selected port.

4. Select an appropriate service operation.

Create Event Subscription - Invoke Web Service: Select Operation

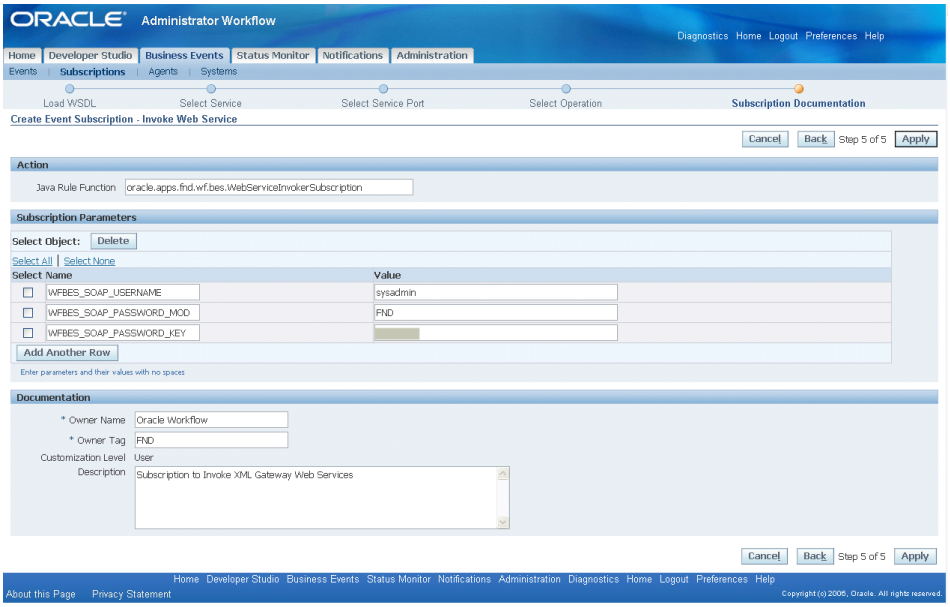


Click **Next** to display the last page of the Create Event Subscription - Invoke Web Service wizard.

5. In the Subscription Documentation of the Create Event Subscription - Invoke Web Service page, the default Java Rule Function name `oracle.apps.fnd.wf.bes.WebServiceInvokerSubscription` is automatically populated.

   Important: If you have extended the functionality of the seeded rule function, manually enter your custom function name here.

Create Event Subscription - Invoke Web Service: Subscription Documentation



6. In the Documentation region, enter an application name or program name that owns the subscription (such as 'Oracle Workflow') in the Owner Name field and the program ID (such as 'FND') in the Owner Tag field. Click **Apply**.

For more information, see Defining Event Subscriptions, *Oracle Workflow Developer's Guide*.

Create an Error subscription with 'Launch Workflow' Action Type

To enable the error processing feature during the service invocation, you must create an Error subscription to the invoker business event.

Once subscribing to this error processing, if any error occurs during the invocation, the error process sends a workflow notification to SYSADMIN. This information includes web service definition, event details, and error details allowing SYSADMIN to easily identify the error. The notification also provides an option for SYSADMIN to respond to the error. The SYSADMIN can invoke the web service again after the underlying issue that caused the error is resolved, abort the errored event if needed, or reassign an errored notification to another user if appropriate.

For detailed information on managing errors during web service invocation, see Managing Errors, page 12-37.

To create an error subscription with 'Launch Workflow' action type:

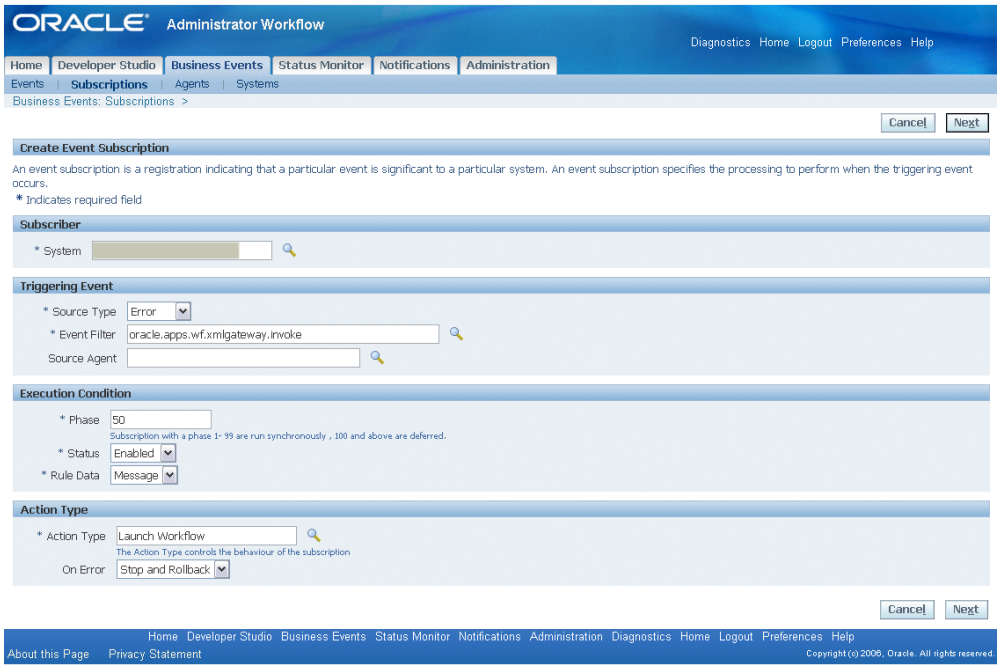
1. Log on to Oracle E-Business Suite with the Workflow Administrator Web responsibility. Select the Business Events link, and choose Subscriptions in the horizontal navigation.

2. In the Event Subscriptions page, click **Create Subscription** to open the Create Event Subscription page.

3. Enter the following information in the Create Event Subscription page:

   - Subscriber: Select the local system
   - Source Type: Error
   - Event Filter: Select the event name that you just created, such as `oracle.apps.wf.xmlgateway.invoke`
   - Phase: this can be any phase number
   - Status: Enabled
   - Rule Data: Key
   - Action Type: Launch Workflow
   - On Error: Stop and Rollback

Create Event Subscription Page



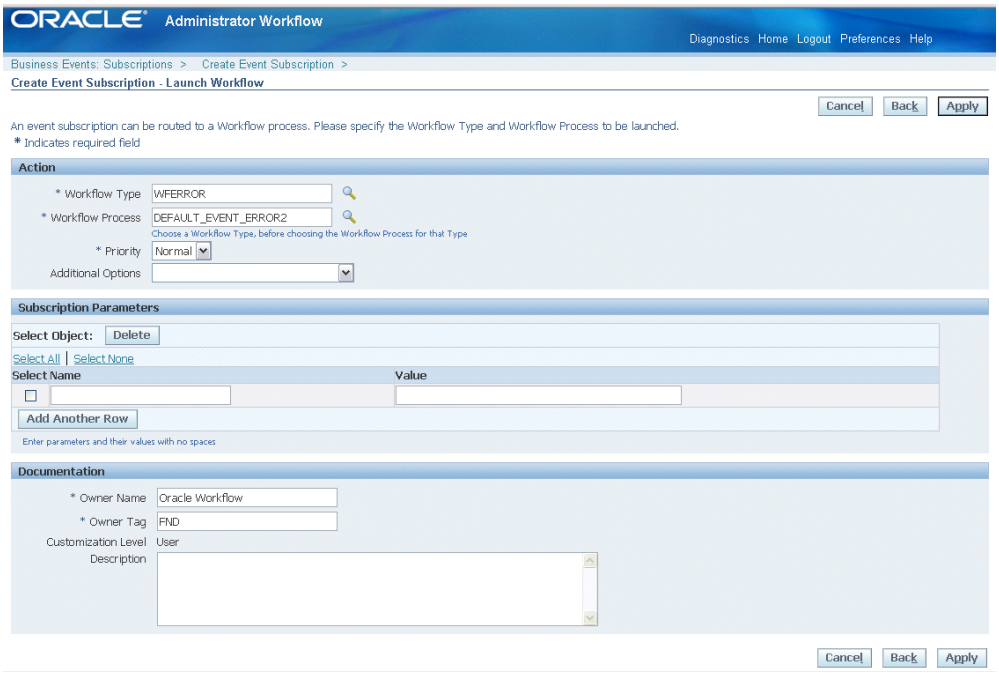
4. Click **Next** to open the Create Event Subscription - Launch Workflow page.

5. Enter the following information in the Action region:

   - Workflow Type: WFERROR
   - Workflow Process: DEFAULT_EVENT_ERROR2
   - Priority: Normal

6. In the Documentation region, enter an application or program name that owns the event subscription (such as Oracle Workflow) in the Owner Name field and application or program ID (such as 'FND') in the Owner Tag field.

Create Event Subscription - Launch Workflow Page



7. Click **Apply**.

Step 3: Creating a Receive Event and Subscription (Optional)

A receive event can serve as a communication vehicle to communicate or callback to Oracle E-Business Suite if a web service has an output or response message required to be communicated back after the web service is successfully invoked. However, whether you need to create a receive event and external subscription to the receive event depends on the following criteria:

- Your message pattern
- Where your event is raised from (Java or PL/SQL layer)
- Event subscription phase number

For Synchronous Request-Response Web Service Invocation

- If the web service invoker event is raised from Java code in the application tier, and the invoker subscription is synchronous with subscription phase < 100, then the web service is invoked as soon as the event is raised, and if successful the response can be read by the calling application and is available immediately by using `BusinessEvent.getResponseData()` method after calling `BusinessEvent.raise()`.

In this case, the response may not have to be communicated back to Oracle E-Business Suite using callback event. Hence, you may not need to create a receive event and the subscription to the event.

- If the web service invoker event is raised from Java code with the subscription phase is >= 100, or if the event is raised from PL/SQL, the event message will be enqueued to WF_JAVA_DEFERRED queue. In this situation, you will need to create a receive event and external subscription to the event if the web service has an output or a response message. Callback event with callback agent is required to receive the output message into Oracle E-Business Suite.

  This receive event can also be used as a callback into Oracle E-Business Suite to let the interested parties know through raising this event that the web service response is available.

  See: Calling Back to Oracle E-Business Suite With Web Service Response, page 12-29.

  If a receive event is required, after creating the receive event, you must create an external event subscription to the receive event. The web service response message communicated through the receive event is always enqueued to an inbound workflow agent. In order to process an event from the inbound workflow agent, an external subscription is required.

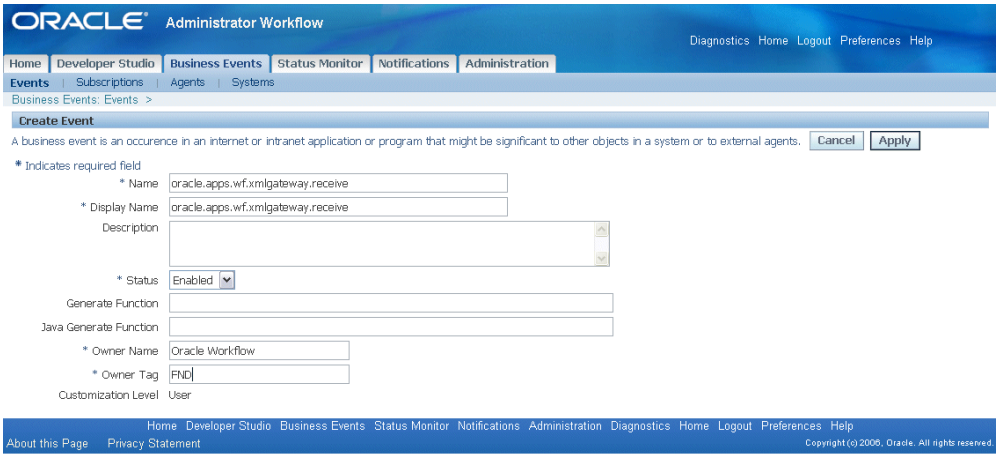
For Request-only Web Service

If it is a request-only web service which does not require a response, you do not need to create a receive event.

To create a receive event:

1. In the Events page, click **Create Event** to open another Create Event page.
2. Enter the following information in the Create Event page:
   - Name: Enter an event name, such as `oracle.apps.wf.xmlgateway.receive`
   - Display Name: Enter an event display name, such as `oracle.apps.wf.xmlgateway.receive`
   - Description: Enter a description for the event
   - Status: Enabled
   - Owner Name: Enter an application or program name that owns the event (such as 'Oracle Workflow')
   - Owner Tag: Enter the application or program ID that owns the event (such as 'FND')

Create Event Page



3. Click **Apply** to create a receive event.

To create a receive event subscription:

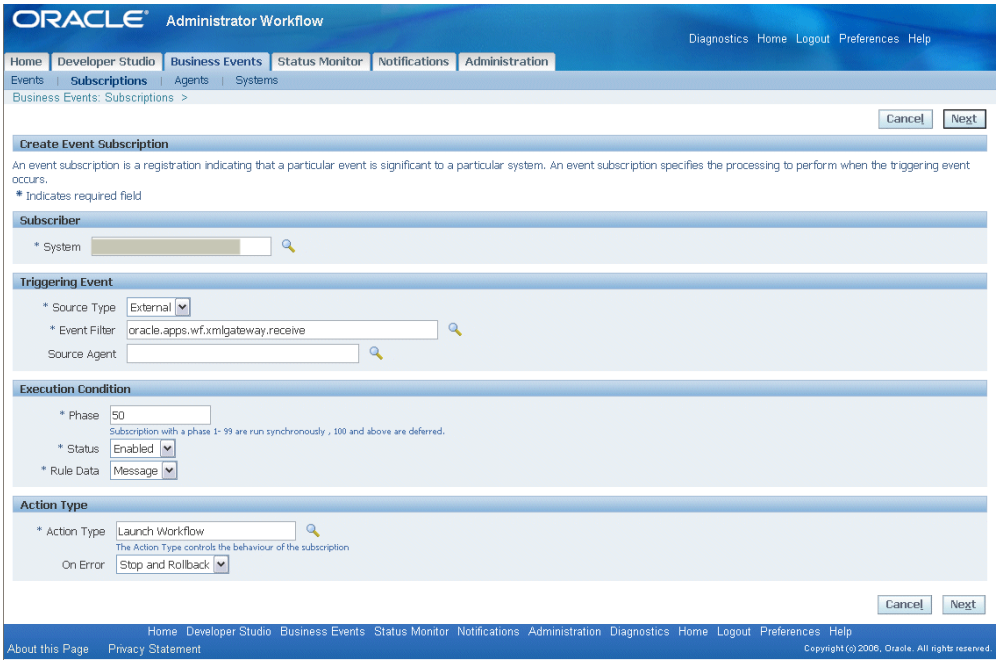
1. Log on to Oracle E-Business Suite with the Workflow Administrator Web Applications responsibility. Select the Business Events link, and choose Subscriptions in the horizontal navigation.

2. In the Event Subscriptions page, click **Create Subscription** to open the Create Event Subscription page.

3. Enter the following information in the Create Event Subscription page:

   - Subscriber: Select the local system
   - Source Type: External
   - Event Filter: Select the receive event name that you just created, such as `oracle.apps.wf.xmlgateway.receive`
   - Phase: any phase number
   - Status: Enabled
   - Rule Data: Key
   - Action Type: any action type
   - On Error: Stop and Rollback

Create Event Subscription Page



4. Click **Next** to open the Create Event Subscription - Launch Workflow page.

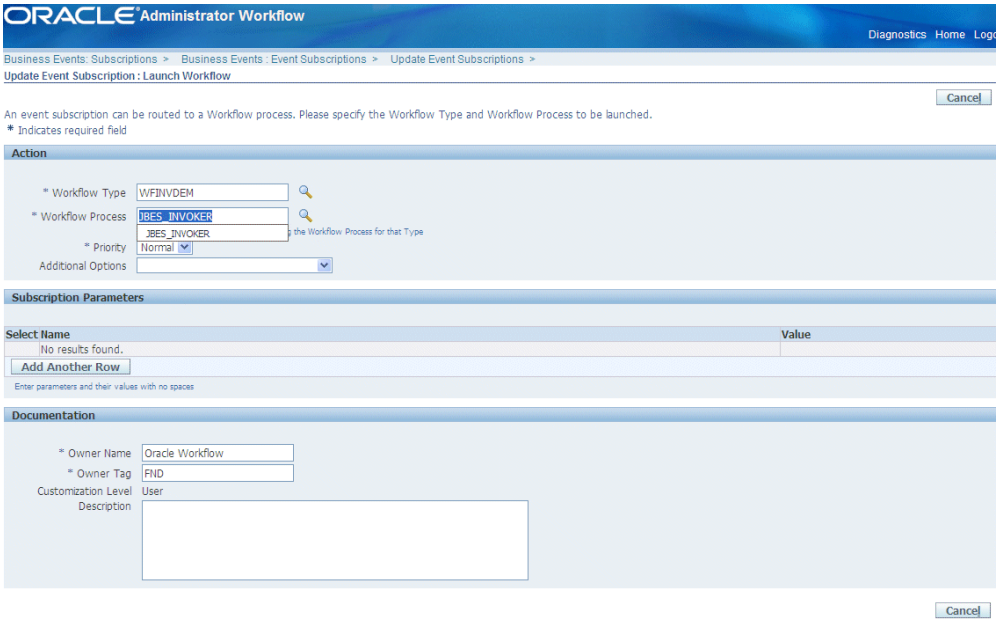
   Note that the type of the Create Event Subscription page to be shown depends on the value selected in the Action Type field. If "Launch Workflow" is selected, you will see the Create Event Subscription - Launch Workflow page. If other action types are selected, different types of the create event subscription pages are displayed. By entering an appropriate action type through the subscription page, you can launch a workflow process or just execute a custom rule function for the event defined as part of this subscription.

5. Enter the following information in the Action region:

   - Workflow Type: Enter any workflow type that is waiting for the response
   - Workflow Process: Enter any workflow process that is waiting for the response
   - Priority: Normal

6. In the Documentation region, enter an application or program name in the Owner Name field (such as 'Oracle Workflow') and application or program ID in the Owner Tag field (such as 'FND').

Update Event Subscription: Launch Workflow Page



7. Click **Apply**.

Understanding Web Service Input Message Parts

A message consists of one or more logical parts. Each part describes the logical abstract content of a message. For example, a typical document-style web service could have a header and body part in the input message.

For example, consider the operation `PROCESSPO` in Oracle E-Business Suite XML Gateway service (`http://<hostname>:<port>/webservices/SOAProvider/xmlgateway/ont__poi/?wsdl`) as described below.

```
<definitions targetNamespace="ONT__POI" targetNamespace="http://xmlns.
oracle.com/apps/ont/soaprovider/xmlgateway/ont__poi/">
<type>
   <schema elementFormDefault="qualified" targetNamespace="http://xmlns.
oracle.com/apps/ont/soaprovider/xmlgateway/ont__poi/">
    <include schemaLocation="http://<hostname>:
<port>/webservices/SOAProvider/xmlgateway/ont__poi/PROCESS_PO_007.xsd"/>
   </schema>
...
<message name="PROCESSPO_Input_Msg">
  <part name="header" element="tns:SOAHeader"/>
   <part name="body" element="tns1:PROCESS_PO_007"/>
</message>
...
<binding name="ONT__POI_Binding" type="tns:ONT__POI_PortType">
<soap: binding style="document" transport="http://schemas.xmlsoap.
org/soap/http"/>
  <operation name="PROCESSPO">
  <soap:operation soapAction="http://host:
port/webservices/SOAProvider/xmlgateway/ont__poi/"/>
  <input>
   <soap:header message="tns:PROCESSPO_Input_Msg" part="header" use="
literal"/>
   <soap:body parts="body" use="literal"/>
   </input>
  </operation>
</binding>
...
</definitions>
```

The operation PROCESSPO requires input message PROCESSPO_Input_Msg, which has two parts:

- Body: The value of PROCESS_PO_007 type to be set as SOAP body is sent as business event payload.
- Header: The value of SOAHeader type to be sent in the SOAP header which is required for web service authorization.

To better understand the web service operation's input message, the section includes the following topics:

- Event Payload as SOAP Body, page 12-23
- Other Web Service Input Message Parts, page 12-26

Event Payload as SOAP Body

Any detail information needed to describe what occurred in an event, in addition to the event name and event key, is called the event data. For example, the event data for a purchase order event includes the item numbers, descriptions, and cost.

During the event creation, you can have the event data specified either with or without using the Generate Function for an event from both PL/SQL and Java. If the application where the event occurs does not provide event data, then you can use the Generate Function while creating the event. The Generate Function will produce the complete event data from the event name, event key, and an optional parameter list at the event raise. Otherwise, you do not need to specify the Generate Function field if the

application where the event occurs does provide event data. In other words, the event payload can be passed in either one of the following ways:

- Event data or payload is passed through the Generate Function during the event raise.

- Event data or payload is passed along with the event itself without using the Generate function.

> **Note:** The generate function must follow a standard PL/SQL or Java API. See *Oracle Workflow Developer's Guide* and *Oracle Workflow API Reference*.

The event data can be structured as an XML document and passed as SOAP body during the event raise. The seeded Java rule function accepts this SOAP body through business event payload. The SOAP body is described in a well-formed XML element that would be embedded into SOAP envelope.

- `BusinessEvent.setData(String)`
- `WF_EVENT.Raise(... p_event_data => ...);`

Message Transformation Parameters to Support XSL Transformation

If the invoker event's XML payload (to be used as web service input message) requires to be transformed into a form that complies with the input message schema, the seeded Java rule function could perform XSL transformation on the payload before invoking the web service. Similarly, if the web service output message requires to be transformed into a form that is required for processing by Oracle E-Business Suite, the seeded Java rule function could perform XSL transformation on the response before calling back to Oracle E-Business Suite.

> **Note:** An input message is the XML payload that is passed to the web service in the SOAP request. An output message is the XML document received as a response from the web service after a successful invocation.

For the synchronous request - response operation, when the output (response) message, an XML document, is available, if this XML document requires to be transformed to a form that is easier for Oracle E-Business Suite to understand, then XSL transformation on the output message will be performed.

> **Note:** The XSL filename is given based on the format of <File Name>:<Application Short Name>:<Version>.
>
> For example, "`PO_XSL_1_1_2.xsl:FND:1.1`".

The XSL file names are passed to the seeded Java rule function as the following

subscription parameters while creating the subscription to the web service invoker event through the Create Event Subscription - Invoke Web Service wizard:

- WFBES_OUT_XSL_FILENAME: XSL file to perform transformation on the output (response) message

  For example, `WFBES_OUT_XSL_FILENAME=PO_XSL_OUT_2.xsl:FND:1.1`

- WFBES_IN_XSL_FILENAME: XSL file to perform transformation on the input message

  For example, `WFBES_IN_XSL_FILENAME=PO_XSL_IN_2.xsl:FND:1.1`

At runtime, the XSL filenames are passed through the same parameters as event parameters. If event parameters are passed with the same names as the subscription parameters that have been parsed and stored, the event parameter values override the subscription parameter values. For example, the event parameters are passed as follows:

- `BusinessEvent.setStringProperty("WFBES_OUT_XSL_FILENAME", "PO_XSL_OUT_2.xsl:FND:1.1");`

- `BusinessEvent.setStringProperty("WFBES_IN_XSL_FILENAME", "PO_XSL_IN_2.xsl:FND:1.1");`

If `WFBES_OUT_XSL_FILENAME` is null, no outbound transformation will be performed. If `WFBES_IN_XSL_FILENAME` is null, no inbound transformation will be performed.

Loading XSL files to Oracle E-Business Suite

The seeded Java rule function performs the XSL transformation on the input and output messages by using the XML Gateway API, `ECX_STANDARD.perform_xslt_transformation`; therefore, the XSL files for the XSL transformation on input and output messages are loaded to Oracle XML Gateway using the `oracle.apps.ecx.loader.LoadXSLTToClob` loader.

> **Note:** For information on the XSL transformation PL/SQL API, see Execution Engine APIs, *Oracle XML Gateway User's Guide*.

As a result, use the following steps to perform XSL transformation during service invocation:

1. Upload the XSL files to Oracle E-Business Suite using the `oracle.apps.ecx.loader.LoadXSLTToClob` loader in Oracle XML Gateway.

2. Specify the XSL file names (such as `PO_XSL_IN_2.xsl:FND:1.1`) in the event or subscription parameters (WFBES_IN_XSL_FILENAME and WFBES_OUT_XSL_FILENAME) if applicable for XSL transformation on input and output messages.

For example, upload the XSL files to Oracle E-Business Suite as follows:

```
java oracle.apps.ecx.loader.LoadXSLTToClob apps password
<hostname>:<port>:<sid> PO_XSL_IN_2.xsl FND 1.1
```

For more information, see Loading and Deleting an XSLT Style Sheet, *Oracle XML Gateway User's Guide*.

Other Web Service Input Message Parts

Apart from passing the SOAP body part as event payload, service invocation framework also supports passing values for other parts that are defined for the web service operation's input message using the business event parameter with the following format:

WFBES_INPUT_`<partname>`

`<partname>` is same as the part name in the input message definition in WSDL.

For example, the header part for above example is passed to business event as parameter **WFBES_INPUT_`header`** during the invoker event raise. The following code snippet shows the header part that is used to pass username, responsibility, responsibility application, and NLS language elements for Web service authorization:

```
 String headerPartMsg = "<ns1:SOAHeader
xmlns:ns1=\"http://xmlns.oracle.com/xdb/SYSTEM\" " +
            "env:mustUnderstand=\"0\"
xmlns:env=\"http://schemas.xmlsoap.org/soap/envelope/\"> \n" +
         " <ns1:MESSAGE_TYPE>XML</ns1:MESSAGE_TYPE>\n" +
         " <ns1:MESSAGE_STANDARD>OAG</ns1:MESSAGE_STANDARD>\n" +
         " <ns1:TRANSACTION_TYPE>PO</ns1:TRANSACTION_TYPE>\n" +
         " <ns1:TRANSACTION_SUBTYPE>PROCESS</ns1:TRANSACTION_SUBTYPE>\n"
+
         " <ns1:DOCUMENT_NUMBER>xxx</ns1:DOCUMENT_NUMBER>\n" +
         " <ns1:PARTY_SITE_ID>xxxx</ns1:PARTY_SITE_ID>\n" +
         "</ns1:SOAHeader>\n";
businessEvent.setStringProperty("WFBES_INPUT_header", headerPartMsg);
```

> **Note:** This WFBES_INPUT_`<partname>` parameter can only be passed at runtime during the event raise, not through the event subscription. Several constants are defined in the interface `oracle.apps.fnd.wf.bes.InvokerConstants` for use in Java code.

If the web service input message definition has several parts, value for the part that is sent as SOAP body is passed as event payload. Values for all other parts are passed as event parameters with parameter name format WFBES_INPUT_`<partname>`. If the value for a specific input message part is optional to invoke the web service, you still have to pass the parameter with null value so that invoker subscription knows to which part the event payload should be set as SOAP body.

For example, if input message part `myheader` for a web service is optional and does not require a valid value for the invocation to succeed, the event parameter for the input should still be set with null value as follows.

```
businessEvent.setStringProperty("WFBES_INPUT_myheader", null);
```

Supporting WS-Security

Web service security (WS-Security) is a communication protocol providing a means for applying security to web services. It describes enhancements to SOAP messaging to provide quality of protection through message integrity and single message authentication. It also describes how to attach security tokens to SOAP messages to enhance security features.

Service invocation framework supports WS-Security in a general-purpose mechanism for associating security tokens with messages to authenticate web service requests and service invocation from Oracle E-Business Suite.

To accomplish this goal, service invocation framework supports WS-Security through UsernameToken based security.

UsernameToken Based Security

This security mechanism provides a basic authentication for Web service invocation by passing a *username* and an optional *password* in the SOAP Header of a SOAP request sent to the web service provider.

Note that the username/password information discussed in this UsernameToken based security model is the concept of Oracle E-Business Suite username/password.

If the web service being invoked enforces Username/Password based authentication, then the service invocation framework also supports the UsernameToken-based WS-Security header during the service invocation.

> **Note:** SOAP requests invoking the web services should include security header consisting of Username and Plain text password. Encryption is not supported in this release.

Username

The username is a clear text, and its value for the operation is stored in the subscription parameter:

WFBES_SOAP_USERNAME

For example, `WFBES_SOAP_USERNAME =SYSADMIN`

Following sample code describes how the username is stored with the subscription as web service metadata:

```
SERVICE_WSDL_URL=http://<hostname>:<port>/OA_HTML/XMLGatewayWSDL
SERVICE_NAME=XMLGateway SERVICE_PORT=XMLGatewayPort
SERVICE_PORTTYPE=XMLGatewayPortType SERVICE_OPERATION=ReceiveDocument
WFBES_SOAP_USERNAME=SYSADMIN
```

Password

Password is the most sensitive part of the UsernameToken profile. Service invocation framework supports the UsernameToken based WS-Security during service invocation

with username and an optional password with Type PasswordText.

> **Note:** The PasswordText password type is the password written in clear text. There is another password type called 'PasswordDigest' which is a base64-encoded SHA-1 hash value of the UTF8-encoded password and this type of password is not supported in this release.

The password corresponding to the SOAP username is stored in FND vault using a PL/SQL script `$FND_TOP/sql/afvltput.sql`. For example,

```
sqlplus apps/password@db @$FND_TOP/sql/afvltput.sql <Module>
<Key> <Value>
```

- `<Module>` parameter represents the application code such as FND, PO, AR, AP and so on for which the service invocation is being implemented.
- `<Key>` parameter should be uniquely named within the module in consideration for namespace issues. It is the developer's discretion to name the key.
- `<Value>` parameter represents the password value.

The module and key values to retrieve the password corresponding to the SOAP username are passed to the following invoker subscription parameters:

- WFBES_SOAP_PASSWORD_MOD

  For example, `WFBES_SOAP_PASSWORD_MOD=FND`

- WFBES_SOAP_PASSWORD_KEY

  For example, `WFBES_SOAP_PASSWORD_KEY=password`

For example, if you want to store the password value ("*password*") for a web service, it could be stored as follows:

```
sqlplus apps/password@db @$FND_TOP/sql/afvltput.sql FND sysadmin
password
```

At runtime, if event parameters are passed with the same names as the subscription parameters that have been parsed and stored, the event parameter values take precedence over the subscription parameters.

For example, the event parameters are passed as follows:

- `BusinessEvent.setStringProperty("WFBES_SOAP_USERNAME", "SYSADMIN");`
- `BusinessEvent.setStringProperty("WFBES_SOAP_PASSWORD_MOD", "FND" );`
- `BusinessEvent.setStringProperty("WFBES_SOAP_PASSWORD_KEY", "password");`

In summary, use the following steps to pass username and password for WS-Security

during web service invocation:

1. Pass the *username* to event or subscription parameter WFBES_SOAP_USERNAME.
2. Store the password in FND Vault using `$FND_TOP/sql/afvltput.sql` script through appropriate module and key.
3. Pass the module name and key to event or subscription parameters WFBES_SOAP_PASSWORD_MOD and WFBES_SOAP_PASSWORD_KEY.

Calling Back to Oracle E-Business Suite With Web Service Response

As mentioned earlier that Oracle Workflow is the primary process management solution within Oracle E-Business Suite; Oracle Workflow Business Event System, an essential component within Oracle Workflow, provides event and subscription features that help identify integration points within Oracle E-Business Suite. Thus, to successfully invoke Web services from Oracle E-Business Suite requires highly integrated environment with Oracle Workflow.

To support synchronous request - response service operation, if a Web service has an output or a response message, service invocation framework uses the *callback* mechanism in Oracle Workflow to communicate the response message back to Oracle E-Business Suite through the Business Event System.

> **Note:** A synchronous request - response message is a common message exchange pattern in Web service operation where a client asks a service provider a question and then waits for a response before continuing on. For more information, see: Understand Message Patterns, page 12-3.

This callback feature takes the invoker event's event key to enqueue the callback event to the specified inbound agent (the callback agent) for the response. In addition, if a workflow process invokes a Web service using "Raise" event activity and waits for Web service response using "Receive" event activity, the invoker event key should be same as the invoker and/or waiting workflow process's item key so that when callback is performed, the waiting workflow process is correctly identified by `WF_ENGINE.EVENT` API.

By using both the *callback* events and agents, Web service invocation can be integrated back with a waiting workflow process or any other module within Oracle E-Business Suite. Web service invocation uses the following callback subscription or event parameters:

- WFBES_CALLBACK_EVENT

  This parameter can have a valid business event to be raised upon completion of the Web service with the service output message as payload.

  For example, it can be like:

```
WFBES_CALLBACK_EVENT=oracle.apps.wf.myservice.callback
```

- WFBES_CALLBACK_AGENT

  This parameter can have a valid business event system agent to which the event with the service response message as payload can be enqueued.

  > **Important:** This parameter will work only if `WFBES_CALLBACK_EVENT` is not null, otherwise the output message is lost and there is no callback.

  For example, it can be like the default inbound agent (or any other inbound queue) for Web service messages:

  ```
  WFBES_CALLBACK_AGENT=WF_WS_JMS_IN
  ```

  > **Note:** If you have defined custom agents, you can also specify the custom agent name as the parameter value.

  Since Web service output message is enqueued to the inbound agent mentioned in WFBES_CALLBACK_AGENT, it is required to set up a Workflow Agent Listener on the inbound agent (if it is not yet set up) in order to process the callback/receive business event messages.

  > **Note:** Callback event can be used as correlation ID when the response message is enqueued to a callback agent. This helps administrators to create specialized agent listeners on a callback agent to process callback events.
  >
  > For example, if the callback event for a service invocation is `oracle.apps.wf.myservice.callback`, and the callback agent is `WF_WS_JMS_IN`, when this event is enqueued to `WF_WS_JMS_IN` upon a successful service invocation, the event `oracle.apps.wf.myservice.callback` is used as Correlation ID in `WF_WS_JMS_IN` to help create an agent listener to process that event.

At run time, if event parameters are passed with the same names as the subscription parameters that have been parsed and stored, the event parameter values take precedence over subscription parameters. For example, the event parameters are passed as follows:

- ```
  BusinessEvent.setStringProperty("WFBES_CALLBACK_EVENT",
  "oracle.apps.wf.myservice.callback");
  ```

- ```
  BusinessEvent.setStringProperty("WFBES_CALLBACK_AGENT",
  "WF_WS_JMS_IN");
  ```

To use the callback feature during the service invocation, you must create a receive

event and subscribe to the receive event. See: Creating a Receive Event and Event Subscription (Optional), page 12-18.

The better understand how to invoke a Web service, see Example of Invoking a Web Service From a Workflow Process, page 12-34

Invoking Web Services

Oracle Workflow Business Event System is a workflow component that allows events to be raised from both PL/SQL and Java layers. Therefore, the service invocation from Oracle E-Business Suite can be from PL/SQL or Java

Service Invocation from PL/SQL

1. Application raises a business event using PL/SQL API `WF_EVENT.Raise`.

   The event data can be passed to the Event Manger within the call to the `WF_EVENT.Raise` API, or the Event Manger can obtain the event data or message payload by calling the Generate function for the event if the data or payload is required for a subscription.

   > **Note:** See *Oracle Workflow API Reference* for information about `WF_EVENT.Raise` API.

2. Oracle Workflow Business Event System (BES) identifies that the event has a subscription with Java Rule Function `oracle.apps.fnd.wf.bes.WebServiceInvokerSubscription`.

3. The Business Event System enqueues the event message to WF_JAVA_DEFERRED queue. The Java Deferred Agent Listener then dequeues and executes the subscription whose Java rule function invokes the web service.

4. If callback event and agent parameters are mentioned, the web service response is communicated back to Oracle E-Business Suite using the callback information. The Java Deferred Agent Listener process that runs in Concurrent Manager (CM) tier invokes the web service.

Service Invocation from Java

1. Java Application raises a business event using Java method `oracle.apps.fnd.wf.bes.BusinessEvent.raise` either from OA Framework page controller/AMImpl or Java code running on Concurrent Manager tier.

2. Since the event is raised in Java where the subscription's seeded Java Rule Function `oracle.apps.fnd.wf.bes.WebServiceInvokerSubscription` is accessible, whether the rule function is executed inline or deferred is determined by the phase of the subscription.

- If the invoker subscription is created with Phase >= 100, the event is enqueued to WF_JAVA_DEFERRED queue.

- If the invoker subscription is created with Phase < 100, the event is dispatched inline.

  If the event is raised from OA Framework page, the dispatch logic executes (that uses WSIF to invoke the web service) within `OACORE OC4J container`.

  > **Note:** If the web service invoker event is raised from Java code in the application tier, and the invoker subscription is synchronous with subscription phase < 100, then the web service is invoked as soon as the event is raised, and if successful the response can be read by the calling application and is available immediately by using method `BusinessEvent.getResponseData()`.
  >
  > If the event is raised from Java code with the subscription phase is >= 100 or if the event is raised from PL/SQL, the event message will be enqueued to WF_JAVA_DEFERRED queue. If the web service has an output or a response message, callback event with callback agent is required to receive the output message into Oracle E-Business Suite.

The following sample Java code raises a business event that invokes web service and reads the response in the same session:

```
package oracle.apps.fnd.wf.bes;

import java.sql.Connection;

import oracle.apps.fnd.common.AppsLog;
import oracle.apps.fnd.common.Log;
import oracle.apps.fnd.wf.bes.InvokerConstants;
import oracle.apps.fnd.wf.common.WorkflowContext;

public class InvokeWebService {

    static Log mLog;
    static WorkflowContext mCtx;

    public InvokeWebService() {
    }

    public static Connection getConnection(String dbcFile) {
        Connection conn = null;

        System.setProperty("dbcfile", dbcFile);
        WorkflowContext mCtx = new WorkflowContext();

        mLog = mCtx.getLog();
        mLog.setLevel(Log.STATEMENT);
        ((AppsLog)mLog).reInitialize();
        mLog.setModule("%");

        return mCtx.getJDBCConnection();
    }

    public static void main(String[] args)
    {
        BusinessEvent event;
        Connection conn;
        conn = getConnection(args[0]);

        try {
            // Proxyt host and port requires to be set in Java
options
            System.setProperty("http.proxyHost", args[1]);
            System.setProperty("http.proxyPort", args[2]);

            event = new BusinessEvent ("oracle.apps.wf.IrepService.
invoke", "eventKey1");

            // Input XML message for Web Service

            String input = null;
            input = "<ns3:
IntegrationRepositoryService_GetInterfaceFunctionByName xmlns:ns3=\"
http://xmlns.oracle.com/apps/fnd/rep/ws\"> \n"+
<fullMethodName>SERVICEBEAN:
/oracle/apps/fnd/rep/ws/IntegrationRepositoryService:
getInterfaceFunctionByNameSERVICEBEAN:
/oracle/apps/fnd/rep/ws/IntegrationRepositoryService:
getInterfaceFunctionByName</fullMethodName>\n"+
"</ns3:IntegrationRepositoryService_GetInterfaceFunctionByName>";
            event.setData(input);

     String headerPartMsg = "<ns1:SOAHeader
xmlns:ns1=\"http://xmlns.oracle.com/xdb/SYSTEM\" " +
            "env:mustUnderstand=\"0\"
xmlns:env=\"http://schemas.xmlsoap.org/soap/envelope/\"> \n" +
         " <ns1:MESSAGE_TYPE>XML</ns1:MESSAGE_TYPE>\n" +
```

```
" <ns1:MESSAGE_STANDARD>OAG</ns1:MESSAGE_STANDARD>\n" +
         " <ns1:TRANSACTION_TYPE>PO</ns1:TRANSACTION_TYPE>\n" +
         " <ns1:TRANSACTION_SUBTYPE>PROCESS</ns1:
TRANSACTION_SUBTYPE>\n" +
         " <ns1:DOCUMENT_NUMBER>123</ns1:DOCUMENT_NUMBER>\n" +
         " <ns1:PARTY_SITE_ID>4444</ns1:PARTY_SITE_ID>\n" +
         "</ns1:SOAHeader>\n";
businessEvent.setStringProperty("WFBES_INPUT_header",
headerPartMsg);

            event.raise(conn);
            conn.commit();

            Object resp = event.getResponseData();
            if (resp != null) {
                System.out.println(resp.toString());
            }
            else {
                System.out.println("No response received");
            }
        }
        catch (Exception e) {
            System.out.println("Exception occured " + e.
getMessage());
            e.printStackTrace();
        }
    }
}
```

Example of Invoking a Web Service From a Workflow Process

The following example is to invoke a web service through launching a workflow process including the following nodes or activities:

- An invoker business event to invoke a web service.

  For example, INVOKE_SERVICE is an event activity with event action "Raise".

- A receive business event to receive a response or web service output message.

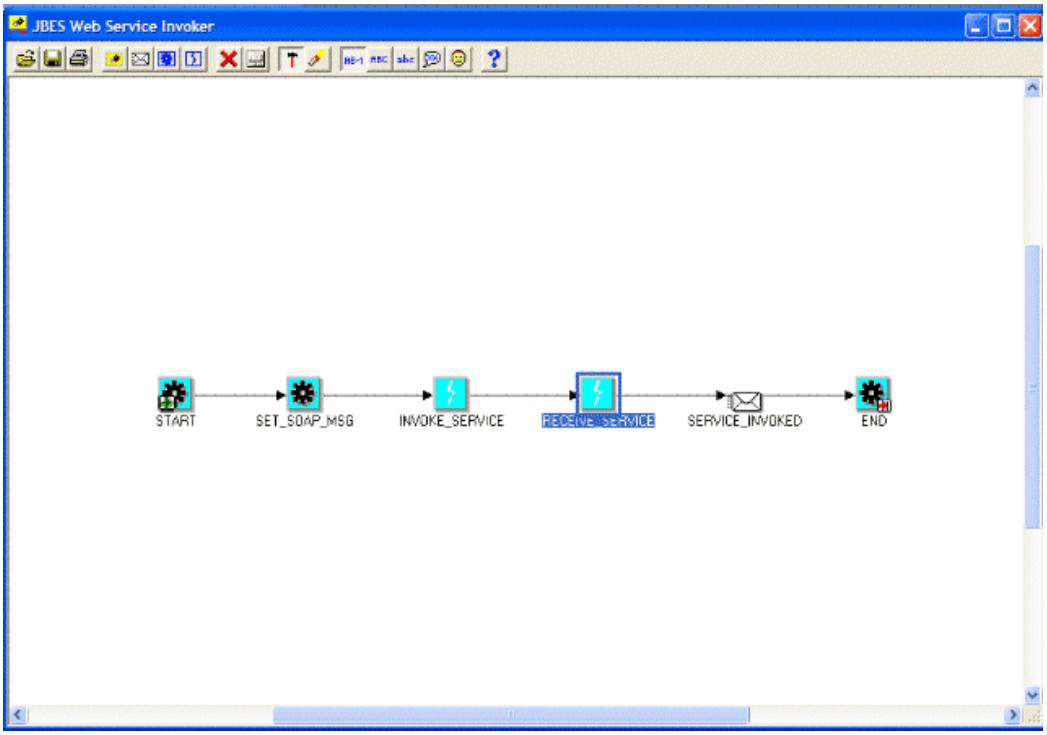
  For example, RECEIVE_SERVICE is an event activity with event action "Receive".

- Other activities could be used in the process for XML message processing, notifying users of web service invocation response, regular transaction processing and so on.

  For example, SERVICE_INVOKED is a notification activity to send a notification message when a web service is successfully invoked.

The following workflow process diagram illustrates the service invocation process flow:

Workflow Process Diagram to Invoke a Web Service



Defining Service Invocation Metadata

To define the service invocation metadata with the callback feature, you must have the following necessary event and subscription in place:

1. An invoker event, such as INVOKE_SERVICE in the workflow diagram.

   This activity is used to pass the event XML payload to be used as SOAP body and other required event parameters required for web service invocation as already discussed.

   See: Creating a Web Service Invoker Business Event, page 12-6.

2. Local and error event subscriptions to the invoker event. See: Creating Local and Error Event Subscriptions to the Invoker Event, page 12-8.

3. A receive event (such as RECEIVE_SERVICE in the workflow diagram) and the External subscription to the receive event.

   Important: The receive event is raised with the same event key as the event key for invoker event. It is important that the waiting workflow process's item key and the invoker event's event key are the same.

If callback event and agent parameters are set, this activity waits for the receive event to occur after web service invocation is successful.

See: Creating a Receive Event and Event Subscription (Optional), page 12-18.

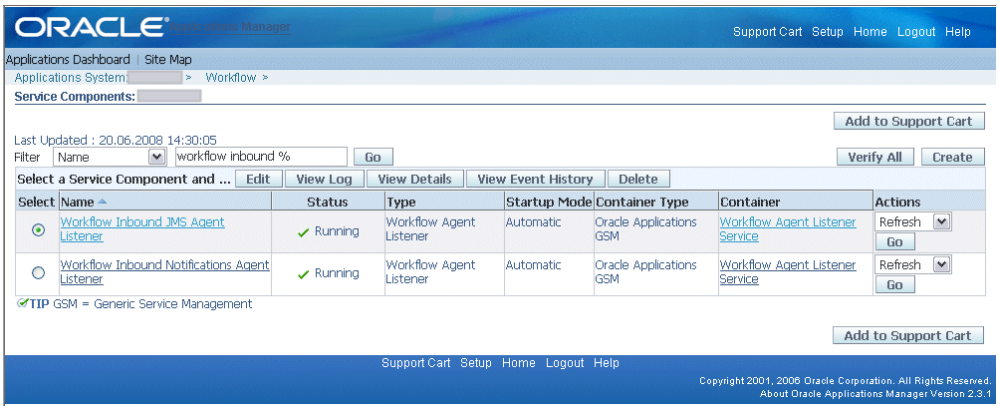
Verifying Workflow Agent Listener Status

In order to process a web service response message from the inbound agent, you need to verify if a Workflow Agent Listener is running on that agent.

Use the following steps for verification:

1. Log on to Oracle Workflow with Oracle Workflow Web Administrator responsibility.
2. From the navigation menu, select Oracle Applications Manager, and click the Workflow Manager link.
3. Click the Agent Listener status icon to open the Service Components page.
4. Locate the Workflow Agent Listener that you use for the callback agent listener. For example, locate the 'Workflow Inbound JMS Agent Listener' for processing a web service response message to ensure it is up and running.

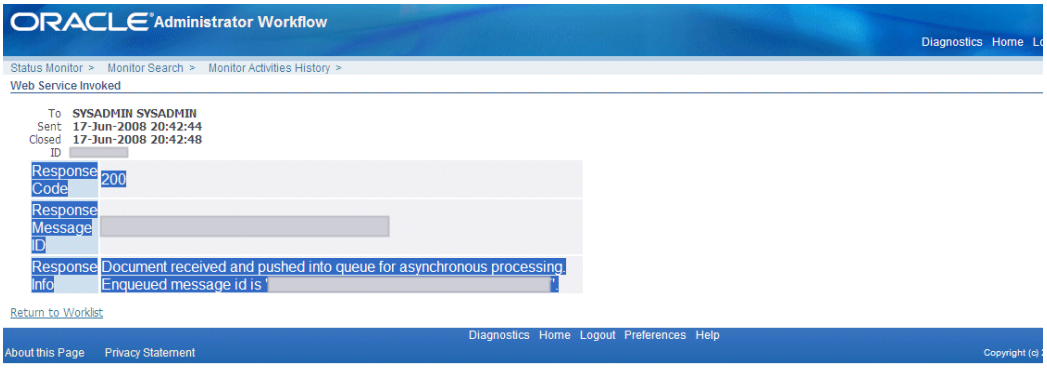
Service Components Page to Validate the Status of a Workflow Agent Listener



After the verification, you can launch the workflow process to invoke a web service with a callback response through Oracle Workflow. You can also validate the process by reviewing the progress status of each activity contained in your workflow process diagram.

When the web service has been successfully invoked from the automated workflow process, you should receive a workflow notification message if the notification activity is included in the process.

Web Service Invoked Page with a Notification Message



For more information on how to create and launch a workflow, see *Oracle Workflow Developer's Guide*.

Managing Errors

Service invocation framework uses the same way of handling errors in Business Event System to manage errors occurred during the execution of business event subscriptions. If the service invocation returns a fault message, the event is enqueued to error queue to trigger error processing. If an exception occurred during invocation process is due to service unavailability, the service faults should be logged and error subscription should be invoked.

To effectively process runtime exceptions for the events that are enqueued to an error queue, service invocation framework uses the following event ERROR process to specifically trigger error processing during the service invocation:

- DEFAULT_EVENT_ERROR2: Default Event Error Process (One Retry Option)

  Note: The DEFAULT_EVENT_ERROR2 Error workflow process is created under WFERROR item type.

For example, if there is a runtime exception when the Workflow Java Deferred Agent Listener executes event subscription to invoke the web service, the event is enqueued to the WF_JAVA_ERROR queue. If the event has an Error subscription defined to launch the Error workflow process `WFERROR:DEFAULT_EVENT_ERROR2`, the Workflow Java Error Agent Listener executes the error subscription which sends a notification to SYSADMIN with web service definition, error details and event details. Since Oracle Workflow default event error handler provides options for SYSADMIN to retry the web service invocation process after verifying that the reported error has been corrected, SYSADMIN can invoke the web service again from the notification if necessary.

However, if there is a runtime exception when invoking the web service by raising the Invoker event with synchronous subscription (phase <100), the exception thrown to the

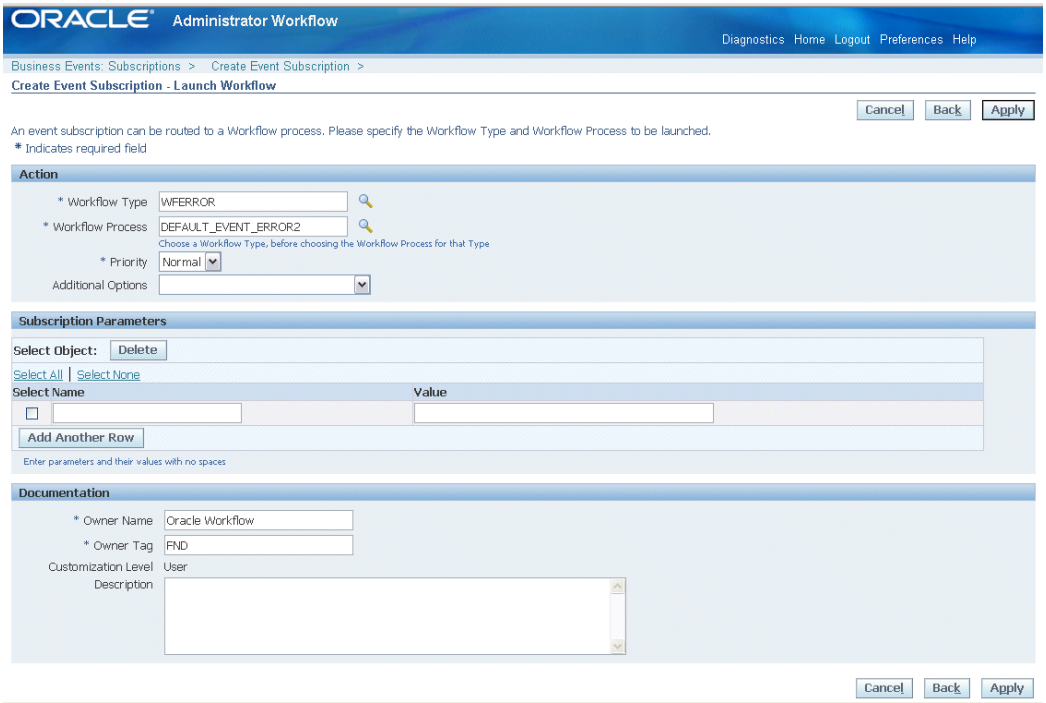
calling application. It is the responsibility of the calling application to manage the exception.

Enabling Error Processing During Service Invocation

To enable the error processing feature during the service invocation, you must create an Error subscription with the following values:

- 'Error' source type
- 'Launch Workflow' action type
- 'WFERROR:DEFAULT_EVENT_ERROR2' workflow process

Create Error Subscription - Launch Workflow Page



To access the Create Event Subscription page, log on to Oracle E-Business Suite with the Workflow Administrator Web Applications responsibility. Select the Business Events link and choose the Subscriptions subtab. In the Event Subscriptions page, click **Create Subscription**.

For information on how to create an error subscription for service invocation, see Create an Error Subscription with 'Launch Workflow' Action Type, page 12-15.

Testing Web Service Invocation

Service invocation framework uses the Oracle Workflow Test Business Event page to

check the basic operation of Business Event System by raising a test event from either Java or PL/SQL layer and executing synchronous and asynchronous subscriptions to that event. This testing feature provides a flexible mechanism which easily lets you validate whether a web service can be successfully invoked from concurrent manager tier and OACORE OC4J.

You can test a web service invocation using one of the following ways:

- Using the Test Business Event Page to Manually Raise an Event, page 12-39
- Using Command Line to Raise an Event, page 12-43

Using the Test Business Event Page

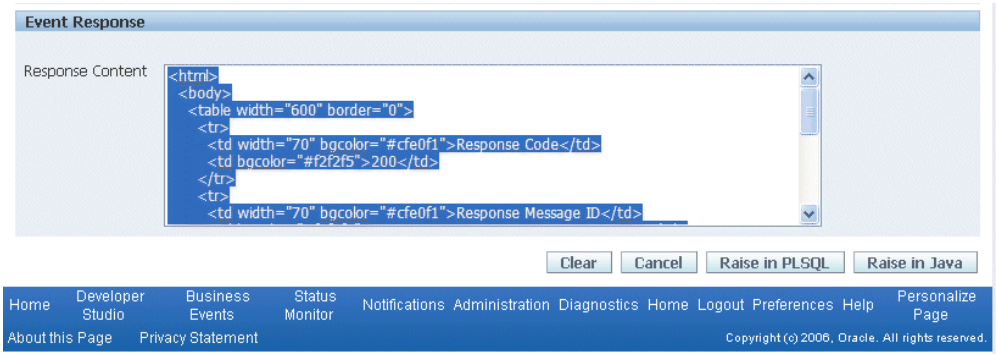
Use the Test Business Event page to test a event by raising it from both PL/SQL API and Java method.

- For an invoker event raised using **Raise in Java** option, the web service is invoked from OACORE OC4J if the subscription phase < 100.

  If the web service is successfully invoked, the Test Business Event page reloads and displays the XML Response region right after the XML Content field.

  If there is a runtime exception when invoking the web service using synchronous subscription, the exception message is shown on the Test Business Event page.

Test Business Event Page: Event Response Region with XML Response Highlighted for a Successful Service Invocation



- For an invoker event raised using **Raise in PLSQL** option, the web service is invoked from the concurrent manager tier. The raised event will be enqueued to WF_JAVA_DEFERRED and then dispatched by Workflow Java Deferred Agent Listener.

  The seeded Java rule function uses the callback event and agent to communicate the response or web service output message back to Oracle E-Business Suite through the Business Event System.

Note: Since Java Deferred Agent Listener is responsible for dispatching the subscription and invoking web services from concurrent manager tier, ensure that Workflow Java Deferred Agent Listener is up and running.

To validate, log on to Oracle Applications Manager and select the **Workflow Manager** link. Choose **Agent Listeners** and search on Workflow Java Deferred Agent Listener to view its status.

Testing Service Invocations

After logging on to Oracle Workflow with the Workflow Administrator Web responsibility. Select the **Business Events** link to search for an event that you want to test. From the search result table, click the **Test** icon next to the event you want to raise. This opens the Test Business Event page where you can raise the event with an unique event key. Enter event parameters for the invoker event subscription and a valid XML message that complies with input message schema. The Test Business Event page will also display response XML message if appropriate.

Note that the Test Business Event page will retain all the data entered. Therefore, if there is a need to raise another event, you must click **Clear** to clear all data that you have entered.

Following parameters may be specified when raising the event from the Test Business Event page to invoke a web service:

- Message transformation: XSL transformation for web service input message and output message
  - WFBES_OUT_XSL_FILENAME
  - WFBES_IN_XSL_FILENAME
- WS-Security: Information required to add UsernameToken header to a SOAP request
  - WFBES_SOAP_USERNAME
  - WFBES_SOAP_PASSWORD_MOD
  - WFBES_SOAP_PASSWORD_KEY
- Input Message part value: Pass values for any part that may be required to embed application context into SOAP envelopes
  - WFBES_INPUT_<partname>

Note: The WFBES_INPUT_<partname> parameter can only be passed at runtime during event raise.

- Callback: Callback to Oracle E-Business Suite with web service response
  - WFBES_CALLBACK_EVENT
  - WFBES_CALLBACK_AGENT
- SOAP Body:
  - XML Input message (Required)

For information about these parameters, see:

- Understanding Web Service Input Message Parts, page 12-22
- Supporting WS-Security, page 12-27
- Calling Back to Oracle E-Business Suite With Web Service Response, page 12-29

Testing Invocation with Callback Required

If you want to test an invocation with callback to Oracle E-Business Suite, then you must enter the following parameters and values:

- WFBES_CALLBACK_EVENT: receive event
- WFBES_CALLBACK_AGENT: WF_WS_JMS_IN (or any other Inbound Queue as the value)

Note that for testing from the Test Business Event page, since the XML message is prewritten and entered in the XML Content field, if there is an error in the input XML message, the error notification will not provide you with an option to correct it before retrying the process.

Before testing the invocation, for easier debugging or troubleshooting purposes throughout the test, you can enable the diagnostics and logging feature to directly display on-screen logs in the test page. For instructions on how to turn on this logging feature, see Troubleshooting Invocation Failures on OACORE OC4J, page 12-44.

To test an event invocation:

1. Log on to Oracle E-Business Suite with the Workflow Administrator Web responsibility and select the Business Events link.
2. Search on a business event that you want to run the test, such as `oracle.apps.wf.xmlgateway.invoke` and click **Go**.

3. Select the business event that you want to raise from the result table and click the **Test** icon to open the Test Business Event page.

4. Enter a unique event key in the Event Key field and leave the Sand Date field blank.

5. Enter appropriate parameters in the Enter Parameters region.

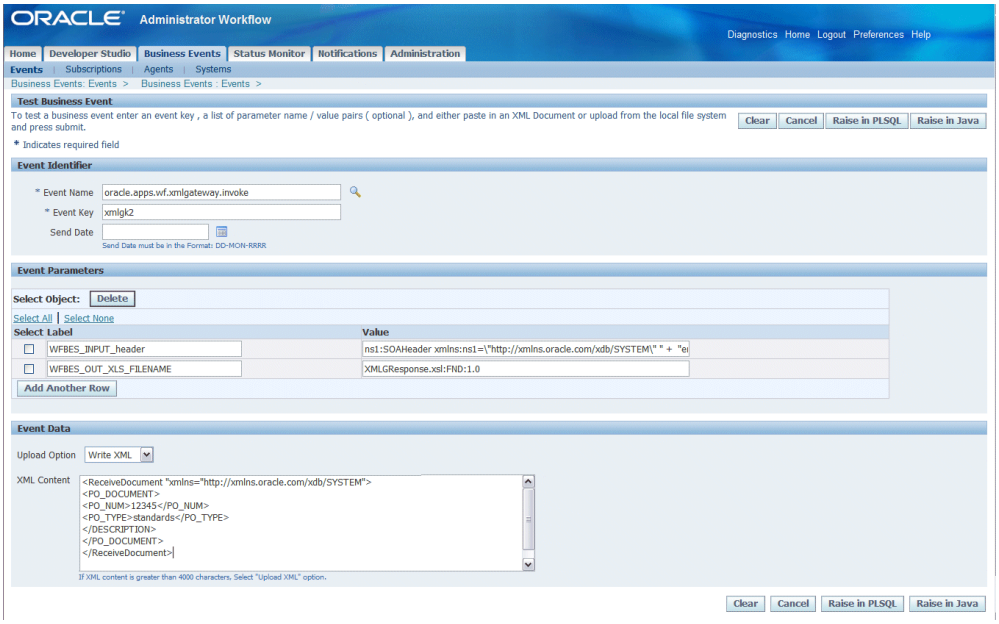
6. In the Event Data region, enter the following information:

   - Upload Option: Write XML

   - XML Content: Enter appropriate XML information as input message. For example, you can enter:

```
<ReceiveDocument xmlns="http://xmlns.oracle.com/xdb/SYSTEM">
   <PO_DOCUMENT>
   <PO_NUM>12345</PO_NUM>
   <PO_TYPE>standards</PO_TYPE>
   </DESCRIPTION>
   </PO_DOCUMENT>
</ReceiveDocument>
```

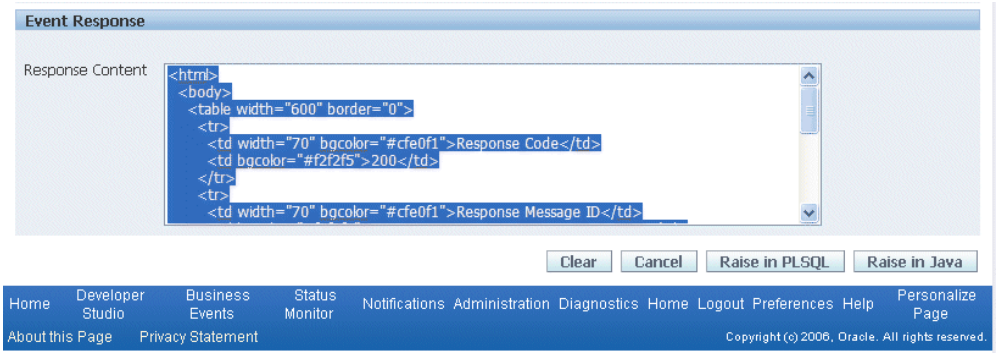
Test Business Event Page



7. Click **Raise in Java** to raise an event from OACORE OC4J.

   If the web service is successfully invoked, the Test Business Event page reloads and displays the XML Response region right after the XML Content field.

Test Business Event Page: Event Response Region



8. Click **Raise in PLSQL** to raise an event is from the concurrent manager tier.

For more information about testing business events, see To Raise a Test Event, *Oracle Workflow Developer's Guide*.

Using Command Lines

You can also use the command line API based test method to raise both PL/SQL-based or Java-based events.

- For PL/SQL-based events, use a PL/SQL `WF_EVENT.Raise` API to test web service invocation from the concurrent manager tier. JVM. Java Deferred Agent Listener dispatches the subscription and invokes web services from concurrent manager tier.

  > **Note:** Since Java Deferred Agent Listener is responsible for dispatching the subscription and invoking wb services from concurrent manager tier, ensure that Workflow Java Deferred Agent Listener is up and running.
  >
  > To validate, log on to Oracle Applications Manager and select the **Workflow Manager** link. Choose **Agent Listeners** and search on Workflow Java Deferred Agent Listener to view its status.

- For Java-based web events, use a Java method `oracle.apps.fnd.wf.bes.BusinessEvent.raise` to test web service invocation.

  For example, we could have a test class `oracle.apps.fnd.wf.bes.WFInvokerTestCase` with classpatch set to `$AF_CLASSPATH`.

  ```
  java oracle.apps.fnd.wf.bes.WSInvokerTestCase <DBC file> <proxy
  host> <proxy port>
  ```

Troubleshooting Web Service Invocation Failure

Web services can be invoked from any one of following tiers:

- **OACORE OC4J**: Web service invocations from the OA Framework page using a synchronous event subscription (phase < 100) is executed from within the OACORE OC4J container.

- **Concurrent Manager (CM) Tier JVM**: The following web service invocations are executed from CM tier JVM within Java Deferred Agent Listener that runs within Workflow Agent Listener Service:

  - Invocations from PL/SQL either through synchronous or asynchronous event subscriptions

  - Invocations from Java/OA Framework through synchronous event subscriptions

- **Standalone JVM**: Web service invocations from a Java process that runs outside OACORE or CM using a synchronous event subscription executes from within that JVM.

In most cases, the web service resides outside the firewall and the executing host does not have direct access to the WSDL or the web service endpoint to send the SOAP request. Without properly setting up and configuring the proxy parameters for each tier that web service invocations occur, WSDL files will not be parsed and consumed during subscription or web services will not be successfully invoked.

For information on setting up proxy host and port appropriately at each layer, see Setup Tasks, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

At runtime, if a web service invocation fails, an exception is thrown and the invoker event is enqueued to the WF_ERROR queue. Since the web service can be invoked from any one of the layers described earlier, how to troubleshoot and resolve the failure invocation can be explained as follows based on the layer that web service invocations occur:

- Troubleshooting Invocation Failure on OACORE OC4J, page 12-44

- Troubleshooting Invocation Failure on Concurrent Manager (CM) Tier JVM, page 12-48

- Troubleshooting Invocation Failure on Standalone JVM, page 12-48

Troubleshooting Invocation Failure on OACORE OC4J

For the purposes of easier debugging or troubleshooting throughout a test run of the web service invocation from within an OA Framework page, on-screen logging

mechanism should be used.

Enabling On-screen Logging

You can enable the on-screen logging feature and have the logs directly displayed at the bottom of the Test Business Event page. These logs provide processing details while executing the code to invoke the web service.

If there is a fault or a runtime exception in processing the event and invoking the service, the on-screen logging quickly discloses what is happening.

Enabling on-screen logging involves the following two steps:

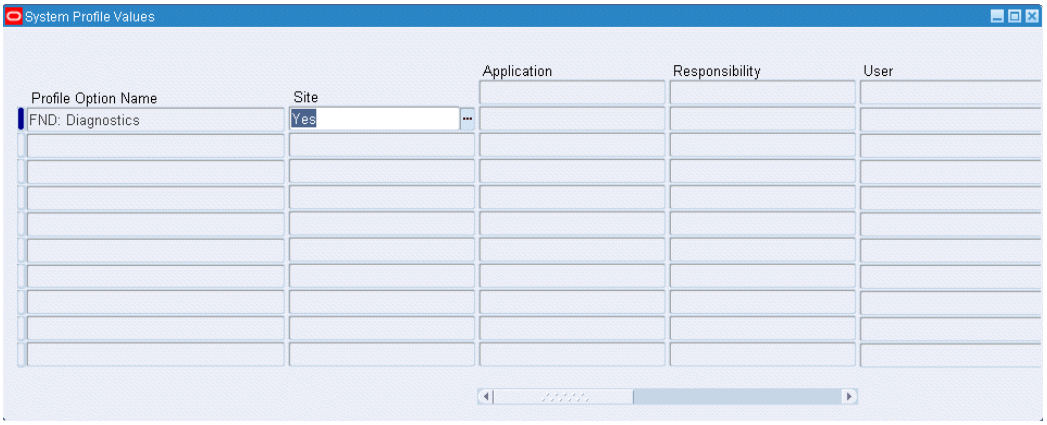
1. Setting FND: Diagnostics Profile Option, page 12-45
2. Displaying On-screen Logging, page 12-45

Setting FND: Diagnostics Profile Option

Before using the Test Business Event page, first set the *FND: Diagnostics* profile option to 'Yes' at an appropriate level to enable the Diagnostics link on the global menu of the HTML-based application pages.

> **Note:** Through the Diagnostics link, we can enable database trace, profiling, and on-screen logging that will help troubleshooting the transactions performed from the HTML-based application pages.

System Profile Values Page to Set the "FND: Diagnostics" Profile Option



With the diagnostics feature, the on-screen logging can be enabled which helps us track the WebServiceInvokeSubscription's log messages when an invoker event is raised from the Test Business Event page and subsequently the web service is invoked.

Displaying On-screen Logging

After setting the "FND: Diagnostics" profile option to 'Yes', you should find the Diagnostics link available in the upper right corner of your HTML page.

By selecting the Diagnostics link and entering appropriate information, the on-screen logging feature can be enabled. Once you locate a desired event and test its invocation, relevant log messages directly appear at the bottom of your test page for an easier debugging or troubleshooting if needed.

> **Important:** If the "FND: Diagnostics" profile option is not set to 'Yes', then the Diagnostics link will not be visible as a global menu for selection. See: Setting FND: Diagnostics Profile Option, page 12-45.

To display on-screen logs while testing your service invocation in the Test Business Event page:

1. Log on to Oracle Workflow with appropriate responsibility, and select the Business Events link to locate an invoker business event hat you want to run the test, such as `oracle.apps.wf.xmlgateway.invoke` and click **Go** to perform a search.

2. From the search result table, select the business event that you want to raise and click the **Test** icon to open the Test Business Event page.

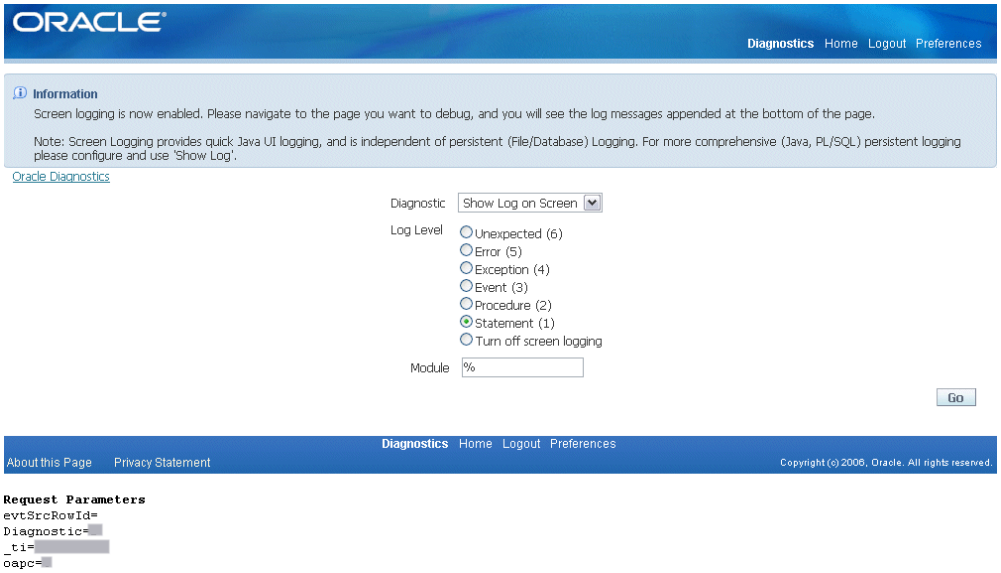
3. Click the Diagnostics link in the upper right corner of the page.

4. Enter the following information to enable the on-screen logs:

   - Diagnostics: Show Log on Screen
   - Log Level: Statement (1)
   - Module: %

5. Click **Go**. The on-screen logging is now enabled.

Oracle Diagnostics Page



6. Navigate to the Test Business Event page and raise an event to execute the invocation testing.

Review On-Screen Log Messages

After you have enabled the on-screen logging feature, during the testing, you should find relevant log messages displayed at the bottom of the Test Business Event page. This provides the detailed information of all processing by the code that invokes the web service.

For example, you can review WebServiceInvokerSupscription log messages displayed on the same page to verify the service execution status, exception or fault if there is any, and whether the callback succeeded or not.

The following example log indicates that the service execution is completed with callback response message enqueued to the WF_WS_JMS_IN inbound queue if the 'WFBES_CALLBACK_EVENT' parameter value is set to receive event and the 'WFBES_CALLBACK_AGENT' parameter value is set to 'WF_WS_JMS_IN':

WebServiceInvokerSubscription Logs

```
5351]:PROCEDURE:[fnd.wf.bes.QueueHandlerInvoker]:enqueue() BEGIN
5351]:PROCEDURE:[fnd.wf.bes.PLSQLQueueHandler]:enqueue() :BEGIN (BusinessEvent(name=Receive event, key=002, priority=1, corre
5351]:EXCEPTION:[fnd.wf.bes.PLSQLQueueHandler]:prepareEnqueueStatement() : Successfully prepared enqueue statement.  Call={ (
5749]:EXCEPTION:[fnd.wf.bes.PLSQLQueueHandler]:enqueue() : Enqueued the following BusinessEvent -> BusinessEvent(name=Receive
5750]:EXCEPTION:[fnd.wf.bes.QueueHandlerInvoker]:enqueue() Enqueued the following BusinessEvent -> BusinessEvent(name=Receive
5750]:STATEMENT:[fnd.wf.bes.WebServiceInvokerSubscription.performCallback]:Enqueued response to WF_WS_JMS_IN
5750]:PROCEDURE:[fnd.wf.bes.WebServiceInvokerSubscription.performCallback]:END
5750]:PROCEDURE:[fnd.wf.bes.WebServiceInvokerSubscription.postInvokeService]:END
5750]:STATEMENT:[fnd.wf.bes.WebServiceInvokerSubscription.onBusinessEvent()]:Service invocation complete
5750]:PROCEDURE:[fnd.wf.bes.WebServiceInvokerSubscription.onBusinessEvent()]:END
```

For detailed information on how to enable the logging feature, see Enabling On-Screen Logging, page 12-44.

Troubleshooting Invocation Failure on Concurrent Manager (CM) Tier JVM

To troubleshoot web service invocation failure on Concurrent Manager (CM) Tier JVM, you must ensure that the Error subscription is created for the all web service invoker events to capture complete exception details when invocation happens from Workflow Java Deferred Agent Listener.

Error Subscription

For all web service invoker events, error subscription is required to enable error processing in the Business Event System that is used to communicate with the SYSADMIN user of an error condition in subscription execution. It sends a workflow notification to the SYSADMIN user with web service definition, error details, and event details allowing the SYSADMIN user to process the errors if needed.

For example, if an error occurs during the invocation and the event is enqueued to the WF_JAVA_ERROR queue, with an Error subscription defined to launch Error workflow process WFERROR:DEFAULT_EVENT_ERROR2, the Workflow Java Error Agent Listener executes the error subscription which sends a notification to the SYSADMIN user with web service definition, error details and event details.

For more information, see Managing Errors, page 12-37.

Enabling Workflow Java Deferred Agent Listener Logging

Since Oracle Workflow default event error handler provides options for the SYSADMIN user to retry the web service invocation process after verifying that the reported error has been corrected, the SYSADMIN user can invoke the web service again from the notification if necessary. However, if further analysis of the steps leading to the exception is required, use Workflow Java Deferred Agent Listener logging mechanism to set the STATEMENT level log for Workflow Java Deferred Agent Listener and retry the failed web service invocation to obtain detailed steps leading to the exception.

For more information, see Java Agent Listeners, *Oracle Workflow Administrator's Guide*.

Troubleshooting Invocation Failure on Standalone JVM

When invoking a web service from a Java process that runs outside OACORE or CM by calling the `BusinessEvent.raise` method to raise the invoker event with a synchronous 'Invoke Web Service' subscription, the following situation can occur:

- If the invocation is successful, the method returns the response message.
- If there was a runtime exception, BusinessEventException, thrown by the method that could be used to get the complete stack trace.

  For details, see the sample Java code in the Service Invocation from Java section, Invoking Web Services, page 12-31.

Extending Seeded Java Rule Function

Oracle E-Business Suite Integrated SOA Gateway allows developers to extend the invoker subscription seeded rule function `oracle.apps.fnd.wf.bes.WebServiceInvokerSubscription` using Java coding standards for more specialized processing.

Developers can extend the seeded rule function to override following methods:

- preInvokeService
- postInvokeService
- invokeService
- addWSSecurityHeader
- setInputParts

For detailed information about these methods, see *Oracle Workflow API Reference*.

preInvokeService

This method is used for pre processing before Web service invocations.

```
protected String preInvokeService(Subscription eo,
                                  BusinessEvent event,
                                  WorkflowContext context)
throws BusinessEventException;
```

The Web service input message or request message is available by calling `event.getData()`. This is the business event payload passed when raising the invoker event or generated by business event Generate function.

This method can perform additional processing on the request data if required. The default implementation through the seeded Java rule function performs XSL transformation using the XSL file specified in `WFBES_IN_XSL_FILENAME` if input payload message is available.

postInvokeService

```
protected void postInvokeService(Subscription eo,
                                 BusinessEvent event,
                                 WorkflowContext context,
                                 String requestData,
                                 String responseData)
throws BusinessEventException;
```

If the operation is synchronous request - response, the response is available in parameter `responseData`.

This method performs additional processing on the response and update application state if required. The default implementation through seeded Java rule function

performs the following tasks:

- XSL transformation on a response or Web service output message based on WFBES_OUT_XSL_FILENAME
- Call back to Workflow Business Event System based on WFBES_CALLBACK_EVENT and WFBES_CALLBACK_AGENT parameter values

invokeService

```
 protected String invokeService(String wsdlLocation,
                                String serviceName,
                                String invokePort,
                                String portTypeName,
                                String operationName,
                                      String eventData)
throws Exception;
```

This invokeService method provides the implementation that makes use of Web service invocation metadata and invokes the Web service using WSIF APIs. This method can be overridden if a different implementation is required other than WSIF to invoke Web services.

addWSSecurityHeader

```
protected void addWSSecurityHeader(ArrayList headersList) throws
Exception;
```

This method adds WS-Security compliant header to the SOAP request. The default implementation through Java seeded rule function adds *UsernameToken* element to the security header based on event parameters WFBES_SOAP_USERNAME, WFBES_SOAP_PASSWORD_MOD, and WFBES_SOAP_PASSWORD_KEY. This method can be overridden to add any WS-Security header or have custom logic to retrieve username and password to build UsernameToken element. The well-formed XML Element should be added to the ArrayList.

The following code snippet shows WS-Security added to a SOAP header:

```
  try {
          DocumentBuilderFactory factory = DocumentBuilderFactory.
newInstance();
          factory.setNamespaceAware(true);
          DocumentBuilder bldr = factory.newDocumentBuilder();
          Document doc = bldr.newDocument();

          Element sec = doc.createElement("wsse:Security");
          Attr attr = doc.createAttribute("xmlns:wsse");
          attr.setValue("http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wsswssecurity-secext-1.0.xsd");
          sec.setAttributeNode(attr);
          doc.appendChild(sec);

          Element unt = doc.createElement("wsse:UsernameToken");
          sec.appendChild(unt);
   .... build XML message ....
        }
        catch (Exception e) {
  }
 headersList.add(doc.getDocumentElement());
```

setInputParts

```
protected void setInputParts(WSIFMessage inputMessage,Input input,String
eventData) throws Exception
```

This `setInputParts` method supports setting input part values such as header, body, or other parts, that are defined for the Web service operation's input message. The default implementation through Java seeded rule function adds the event data payload as the body of the input message. It also adds any other parts provided as event parameters in the triggering event.

The event parameters that contain input message parts must be identified by parameter names with the following format:

`WFBES_INPUT_<partname>`

This method can be overridden to set specific input parts that you require or to set values for RPC (remote procedure call) style Web service invocation.

> **Important:** The service invocation framework supports invoking document-based Web service only. The RPC style Web service invocation is not naturally supported in this release unless you extend this method to set input part values for RPC style.

The following code snippet shows how this method is used to set values for document style Web service invocation:

```
protected void setInputParts(WSIFMessage inputMessage, Input input,
String eventData)
throws Exception {
   BusinessEvent event = this.getBusinessEvent();
   String bindingStyle = this.getBindingStyle();
   if (bindingStyle.equalsIgnoreCase("document") {
      String headerPartMsg = event.getStringProperty
("WFBES_INPUT_header");
      // header part
      inputMessage.setObjectPart("header", getDocumentElement
(headerPartMsg));
      // body part
      inputMessage.setObjectPart("body", getDocumentElement(eventData));
   else {
      // Web service style is RPC
      // Code can be added to set input parts for RPC style invocation
   }
}
```

Sample Codes

The following code shows how to extend the seeded Java rule function:

```
package oracle.apps.fnd.wf.bes;

import java.io.ByteArrayInputStream;

import java.util.ArrayList;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

import oracle.apps.fnd.common.Log;
import oracle.apps.fnd.wf.bes.server.Subscription;
import oracle.apps.fnd.wf.common.WorkflowContext;

import org.apache.wsif.WSIFConstants;
import org.apache.wsif.WSIFMessage;
import org.apache.wsif.WSIFOperation;

import org.w3c.dom.Document;
import org.w3c.dom.Element;


public class MyWebServiceInvoker
  extends WebServiceInvokerSubscription
  implements SubscriptionInterface
{

  private static final String CLASS_PREFIX =
  MyWebServiceInvoker.class.getName() + ".";

  public MyWebServiceInvoker()
   {
   }

  final protected String invokeService( String wsdlLocation,
              String serviceName,
              String invokePort,
              String portTypeName,
              String operationName,
              String eventData)
  throws Exception {
     super.preInvokeService(wsdlLocation, serviceName, invokePort,
          portTypeName, operationName, eventData);

 // Perform special pre invocation processing like updating application
 // state and so on.
   }

 final protected void postInvokeService(Subscription eo,
              BusinessEvent event,
              WorkflowContext context,
              String requestData,
              String responseData)
  throws Exception {

   super.postInvokeService(eo, event, context, requestData,
          responseData);

 // Perform special post invocation processing like updating application
 // state and so on.
   }

  /**
     * Implementing addSOAPHeaders method to include custom header
    * required to invoke an EBS Web Service.
    */
  final protected void addSOAPHeaders(WSIFOperation operation)
```

```
throws Exception {
   String METHOD_NAME = "addSOAPHeaders";
   mLog.write(CLASS_PREFIX + METHOD_NAME, "BEGIN", Log.PROCEDURE);

   WSIFMessage hdrMsg = operation.getContext();
   ArrayList hdr = new ArrayList();

   // Call seeded implementation to add WS-Security header
      super.addWSSecurityHeader(hdr);

   // Add my own Custom header
   mLog.write(CLASS_PREFIX + METHOD_NAME, "Adding Custom header",
   Log.STATEMENT);
   addMyCustomHeader(hdr);

   // Set the headers to WSIFOperation
   hdrMsg.setObjectPart(WSIFConstants.CONTEXT_REQUEST_SOAP_HEADERS,
         hdr);
   operation.setContext(hdrMsg);

   mLog.write(CLASS_PREFIX + METHOD_NAME, "END", Log.PROCEDURE);
    }

  final protected void addMyCustomHeader(ArrayList headersList)
   throws Exception {
    String METHOD_NAME = "addMyCustomHeader";
    // Adding special Custom Header to the WSIF's SOAP request
    String custHdr =
 "ns1:SOAHeader
  xmlns:ns1=\"http://xmlns.oracle.com/xdb/SYSTEM\" " +
 "env:mustUnderstand=\"0\"
  xmlns:env=\"http://schemas.xmlsoap.org/soap/envelope/\"> \n" +
   " <ns1:MESSAGE_TYPE>XML</ns1:MESSAGE_TYPE>\n" +
   " <ns1:MESSAGE_STANDARD>OAG</ns1:MESSAGE_STANDARD>\n" +
   " <ns1:TRANSACTION_TYPE>PO</ns1:TRANSACTION_TYPE>\n" +
   " <ns1:TRANSACTION_SUBTYPE>PROCESS</ns1:TRANSACTION_SUBTYPE>\n" +
   " <ns1:DOCUMENT_NUMBER>xxx</ns1:DOCUMENT_NUMBER>\n" +
   " <ns1:PARTY_SITE_ID>xxxx</ns1:PARTY_SITE_ID>\n" +
    " </ns1:SOAHeader>\n";

   if (custHdr != null && !"".equals(custHdr)) {
            try {
                DocumentBuilderFactory factory =
                DocumentBuilderFactory.newInstance();
                factory.setNamespaceAware(true);
                DocumentBuilder bldr = factory.newDocumentBuilder();
                Document doc = bldr.newDocument();

                doc = bldr.parse(new
                ByteArrayInputStream(custHdr.getBytes()));

                // Add the element to the Headers list
                headersList.add((Element)doc.getFirstChild());
            }
            catch (Exception e) {
                throw new BusinessEventException(
       "Exception when creating header element - "+e.getMessage());
            }
        }
        mLog.write(CLASS_PREFIX + METHOD_NAME, "END", Log.PROCEDURE);
    }

}
```

Other Invocation Usage Considerations

While implementing the service invocation framework to invoke Web services, some limitations need to be considered.

- WFBES_INPUT_<partname> parameter can only be passed at run time during the event raise.

- The service invocation framework supports invoking only document-based Web services.

- Support One-to-One relationship of event subscriptions

  To successfully invoke Web services, each event should only have one subscription (with 'Invoker Web Service' action type) associated with it. This one-to-one relationship of event subscription is especially important in regards to synchronous request - response service invocation.

For detailed information about implementation consideration on service invocation framework, see Implementation Limitation and Consideration, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

A

Integration Repository Annotation Standards

General Guidelines

The Oracle Integration Repository is a centralized repository that contains numerous integration interface endpoints exposed by applications throughout the entire Oracle E-Business Suite. The Integration Repository is populated by the parsing of annotated source code files. Source code files are the "source of truth" for Integration Repository metadata, and it is vitally important that they are annotated in a prescribed and standardized fashion.

This section describes what you should know in general about Integration Repository annotations, regardless of the source code file type that you are working with.

Annotation Syntax

Annotations are modifiers that contain an annotation type and zero or more member-value pairs. Each member-value pair associates a value with a different member of the annotation type.

The annotation syntax is similar to Javadoc syntax:

```
@NameSpace:TypeName keyString
@NameSpace:TypeName freeString
@NameSpace:TypeName keyString keyString keyString
@NameSpace:TypeName keyString freeString
@NameSpace:TypeName {inline annotation} {inline annotation}
```

Element Definitions

`NameSpace` identifies the group of annotations that you are using. It is case sensitive. The annotations currently in use are in the `rep` namespace. Future annotations may be introduced in different namespaces.

`TypeName` identifies the name of the annotation type. It is case sensitive. For

consistency across product teams, always use lowercase typenames.

`keyString` is the first word that follows the annotation. It is a whole string that excludes spaces.

`freeString` is a string that follows the keystring. It may have spaces or inline annotations. It is terminated at the beginning of the next annotation or at the end of the documentation comment.

Format Requirement

In your source code file, repository annotations will appear as a Javadoc-style block of comments.

Use the following general procedure. (If you are working in Java and your file already has robust Javadoc comments, then in many cases you'll only need to add the appropriate "`@rep:`" tags.)

- Choose which interfaces you will expose to the Integration Repository. Be mindful that you can annotate interfaces as *public*, *private*, or *internal*, as well as *active*, *obsolete* , *deprecated*, or *planned*.

  Only interfaces that you annotate as public will appear in the external Integration Repository UI; private and internal interfaces will appear in an internal-only Oracle UI. Consequently, all interfaces that have previously been documented as public in customer manuals should be defined as public in your source file annotations.

- In your source file, set off the beginning of the annotation block according to the following conditional rule:

  - For Java, insert "slash-star-star" characters (`/**`).

  - For non-Java files, insert "slash-star-pound" characters (`/*#`).

- Enter a text description. Use complete sentences and standard English.

- Where applicable, add plain Javadoc tags such as `@param` and `@return`.

- Next, add "`@rep:`" tags such as `@rep:scope` and `@rep:product`.

- Optionally, add a nonpublishable comment using the `@rep:comment` annotation. (Use for reminders, notes, and so on. The parsers skip this annotation.)

- End the annotation block with a "star-slash" (`*/`).

Refer to the following example. Note that the first line could alternatively be slash-star-pound (`/*#`) if the source file was PL/SQL or another non-Java technology.

```
/**
 * This is the first sentence of a description of a sample
 * interface. This description can span multiple lines.
 * Be careful for public interfaces, where the description is
 * displayed externally in the Integration Repository UI.
 * It should be reviewed for content as well as spelling and
 * grammar errors. Additionally, the first sentence of
 * the description should be a concise summary of the
 * interface or method, as the repository UI will display
 * the first sentence by itself.
 *
 * @param <param name> <parameter description>
 @rep:paraminfo {@rep:innertype <typeName>} {@rep:precision <value>}
{@rep:required}
 * @rep:scope <public | internal | private>
 * @rep:product <product short code>
 * @rep:displayname Sample Interface
 */
```

Annotation Syntax Checker and iLDT Generator

A syntax checker is available at the following directory:
`$IAS_ORACLE_HOME/perl/bin/perl $FND_TOP/bin/irep_parser.pl`

Details about the checker can be found by using the `-h` flag.

Class Level vs. Method Level

For the purpose of classifying annotation requirements, we are using loose definitions of the terms "class" and "method". In the context of interface annotations, PL/SQL packages are thought of as classes, and PL/SQL functions or procedures are thought of as methods. For some technologies there are different annotation requirements at the class level and the method level. See the "Required" and "Optional" annotation lists below for details.

Concurrent Program Considerations

In cases where a Concurrent Program (CP) is implemented with an underlying technology that is also an interface type (such as a PL/SQL or Java CP) there may be some confusion as to what needs to be annotated.

Assuming that you intend to have the Concurrent Program exposed by the repository, you should annotate the Concurrent Program. Do not annotate the underlying implementation (such as PL/SQL file) unless you intend to expose it separately from the concurrent program in the repository.

The annotation standards for the following integration interfaces are discussed in this chapter:

- Java Annotations, page A-4
- PL/SQL Annotations, page A-18
- Concurrent Program Annotations, page A-23
- XML Gateway Annotations, page A-25

- Business Event Annotations, page A-35
- Business Entity Annotations, page A-41
- Composite Service - BPEL Annotations, page A-110
- Glossary of Annotations, page A-116

Java Annotations

Users will place their annotations in Javadoc comments, immediately before the declaration of the class or method.

Required Class-level Annotations

- must begin with description sentence(s), page A-116
- rep:scope, page A-119
- rep:product, page A-120
- rep:implementation, page A-121 (only required for Java business service objects; not required for plain Java or SDOs)
- rep:displayname, page A-121
- rep:service, page A-138
- rep:servicedoc, page A-139

Optional Class-level Annotations

- link, page A-125
- see, page A-126
- rep:lifecycle, page A-123
- rep:category, page A-128

  Use BUSINESS_ENTITY at the class level only if all underlying methods have the same business entity. In those cases, you do not need to repeat the annotation at the method level.

  Use IREP_CLASS_SUBTYPE JAVA_BEAN_SERVICES at the class level to indicate that a Java API is serviceable. See Annotations for Java Bean Services, page A-6.

  Use IREP_CLASS_SUBTYPE AM_SERVICES at the class level to indicate that an Application Module class of a Java API is serviceable. See Annotations for

Application Module Services, page A-6.

- rep:compatibility, page A-124
- rep:standard, page A-131
- rep:ihelp, page A-127
- rep:metalink, page A-128
- rep:synchronicity, page A-140

Required Method-level Annotations

- must begin with description sentence(s), page A-116
- param, page A-134

  Use only when applicable and when other tags such as `@see` and `@rep:metalink` do not provide parameter explanations.
- return, page A-135 (if applicable)
- rep:paraminfo, page A-135

  Use parameter level annotation `@rep:paraminfo {@rep:required} {@rep:key_param}` for Java APIs as REST services. See: Annotations for Java Bean Services, page A-6 and Annotations for Application Module Services, page A-6.
- rep:paraminfo, page A-135
- rep:displayname, page A-121
- rep:businessevent, page A-137 (if an event is raised)

Optional Method-level Annotations

- link, page A-125
- see, page A-126
- rep:scope, page A-119
- rep:lifecycle, page A-123
- rep:compatibility, page A-124
- rep:category, page A-128

  Use BUSINESS_ENTITY at the method level only when a class methods have

heterogeneous business entities.

- rep:ihelp, page A-127
- rep:metalink, page A-128
- rep:appscontext, page A-141
- rep:synchronicity, page A-140
- rep:primaryinstance, page A-142
- rep:httpverb, page A-132

  Use this annotation to specify the HTTP Verbs suitable for a Java or an Application Module method. See: Annotations for Java Bean Services, page A-6 and Annotations for Application Module Services, page A-6.

Annotations for Java Bean Services

Not all Java APIs registered in the Integration Repository can be exposed as REST services. Only Java API parameters that are either serializable Java Beans or simple data types such as `String`, `Int`, and so forth can be exposed as Java Bean Services.

In addition to existing Java specific annotations, add the following optional annotations in a .Java file to annotate Java APIs as REST services:

- `@rep:category IREP_CLASS_SUBTYPE JAVA_BEAN_SERVICES`

  This class-level annotation marks a Java API as a serviceable interface.

  It is applicable for .Java files only.

  For more information, see rep:category, page A-128.

- `@rep:httpverb <comma separated list of HTTP VERBS ? GET, POST>`

  This method-level annotation explicitly identifies the HTTP verbs suitable for a method or an operation.

  For more information, see: rep:httpverb, page A-132.

- `@rep:paraminfo {@rep:required} {@rep:key_param}`

  This parameter-level annotation marks path variables.

  `@rep:key_param` is an inline annotation added to an existing `@rep:paraminfo` annotation.

  For more information, see: rep:paraminfo, page A-135.

Annotations for Application Module Services

Similar to Java Bean Services, a system integration developer needs to add the following

optional annotations to annotate Application Module Implementation java class which is a .java file for Application Module Services:

- `@rep:category IREP_CLASS_SUBTYPE AM_SERVICES`

  This class-level annotation marks an Application Module as a serviceable interface.

  It is applicable for .Java files only.

  For more information, see rep:category, page A-128.

- `@rep:httpverb <comma separated list of HTTP VERBS ? GET, POST>`

  As mentioned earlier for the Java Bean Services, this method-level annotation explicitly identifies the HTTP verbs suitable for a method or an operation.

  See: rep:httpverb, page A-132.

- `@rep:paraminfo {@rep:required} {@rep:key_param}`

  Similar to the annotations for Java Bean Services, this parameter-level annotation marks path variables.

  See: rep:paraminfo, page A-135.

Once the system integration developer completes the annotation for the Application Module Services, the annotated interface definition needs to be validated through the Integration Repository Parser (IREP Parser). If no error occurs during the validation, an Integration Repository loader file (iLDT) can be generated. An integration repository administrator can then upload the iLDT file to the Integration Repository using FNDLOAD.

Template

You can use the following template when annotating Application Module Services:

```
Interface Template:

 /**
  * < Interface description
  *   ...
  * >
  *
  * @rep:scope <public>
  * @rep:product <product code>
  * @rep:displayname <Interface display name>
  * @rep:lifecycle <active|deprecated|obsolete|planned>
  * @rep:category IREP_CLASS_SUBTYPE AM_SERVICES
  * @rep:category BUSINESS_ENTITY <business_entity_code>
<sequenceNumber>
  */

Methods Template:

 /**
  * < Method description
  *   ...
  * >
  *
  * @param <paramName> < Parameter description
  *                      ... >
  * @rep:paraminfo {@rep:innertype <typeName>} {@rep:precision <value>}
{@rep:required} {@rep:key_param}
  *
  *
  * @return < Parameter description
  *            ... >
  * @rep:paraminfo {@rep:innertype <typeName>} {@rep:precision <value>}
{@rep:required}
  *
  *
  * @rep:scope <public|private|internal>
  * @rep:displayname <Interface display name>
  * @rep:httpverb <GET|POST|GET,POST>
  * @rep:lifecycle <active|deprecated|obsolete|planned>
  * @rep:category BUSINESS_ENTITY <business_entity_code>
<sequenceNumber>
  */
```

You can use the following template when annotating Java Bean Services:

```
Interface Template:

 /**
  * < Interface description
  *   ...
  * >
  *
  * @rep:scope <public>
  * @rep:displayname <Interface display name>
  * @rep:product <product code>
  * @rep:lifecycle <active|deprecated|obsolete|planned>
  * @rep:category BUSINESS_ENTITY <business_entity_code>
<sequenceNumber>
  * @rep:category IREP_CLASS_SUBTYPE JAVA_BEAN_SERVICES
  */

Methods Template:

 /**
  * < Method description
  *   ...
  * >
  *
  * @param <paramName> < Parameter description
  *                      ... >
  * @rep:paraminfo {@rep:innertype <typeName>} {@rep:precision <value>}
{@rep:required} {@rep:key_param}
  *
  *
  * @return < Parameter description
  *           ... >
  * @rep:paraminfo {@rep:innertype <typeName>} {@rep:precision <value>}
{@rep:required}
  *
  *
  * @rep:scope <public|private|internal>
  * @rep:displayname <Interface display name>
  * @rep:httpverb <GET|POST|GET,POST>
  * @rep:lifecycle <active|deprecated|obsolete|planned>
  * @rep:category BUSINESS_ENTITY <business_entity_code>
<sequenceNumber>
  * @rep:businessevent <businessEventName>
  */
```

You can use the following template when annotating Business Service Objects:

```
Interface Template:

 /**
  * < Interface description
  *   ...
  * >
  *
  * @rep:scope <public|private|internal>
  * @rep:displayname <Interface display name>
  * @rep:lifecycle <active|deprecated|obsolete|planned>
  * @rep:product <product code>
  * @rep:compatibility <S|N>
  * @rep:implementation <full implementation class name>
  * @rep:category <lookupType> <lookupCode> <sequenceNumber>
  */

Methods Template:

 /**
  * < Method description
  *   ...
  * >
  *
  * @param <paramName> < Parameter description
  *                      ... >
  * @rep:paraminfo {@rep:innertype <typeName>} {@rep:precision <value>}
{@rep:required}
  *
  *
  * @return < Parameter description
  *            ... >
  * @rep:paraminfo {@rep:innertype <typeName>} {@rep:precision <value>}
{@rep:required}
  *
  *
  * @rep:scope <public|private|internal>
  * @rep:displayname <Interface display name>
  * @rep:lifecycle <active|deprecated|obsolete|planned>
  * @rep:compatibility <S|N>
  * @rep:category <lookupType> <lookupCode> <sequenceNumber>
  * @rep:businessevent <businessEventName>
  */
```

Examples

Here is an example of an annotated Workflow Worklist Application Module Service:

Class level:

```
 /**
  * This is a Workflow Worklist Application Module Implementation class
  * which provides APIs to set preferred lists, get user worklist,
  * get lists and search the worklist based on certain filter criteria
  * like viewId, status, block size and block sequence.
  * @rep:scope public
  * @rep:product FND
  * @rep:displayname Workflow Worklist
  * @rep:category IREP_CLASS_SUBTYPE AM_SERVICES
  * @rep:category BUSINESS_ENTITY WF_WORKLIST
  */

public class WFWorklistServiceAMImpl extends OAApplicationModuleImpl {
...
```

Method level:

```
    /**
       * This is the method for getting worklist summary for a user
       * @param blockSize Block Size specifies the number of records to
be fetched, cannot be null
    * @paraminfo {@rep:required}
       * @param blockSequence Block Sequence, cannot be null, value >=1
    * @paraminfo {@rep:required}
       * @return Array of notifications
       * @rep:displayname Get HomePage Worklist
       * @rep:httpverb  get, post
       * @rep:category BUSINESS_ENTITY WF_WORKLIST
       */
      public Output[] getHomePGWorklist(String blockSize, String
blockSequence) throws Exception {
...
```

> **Note:** The annotations for parameters `@param` and `@paraminfo` should be in the same sequence as defined in the method or procedure signature.

Here is an example of an annotated Employee Information service:

```
package oracle.apps.per.sample.service;

...

/**
 * A sample class to demonstrate how Java API can use the ISG REST
framework. This class provides
 * methods to retrieve list of direct reports, all reports of a person.
It also has methods to
 * retrieve personal details and accrual balance of a person.
 * @rep:scope public
 * @rep:product PER
 * @rep:displayname Employee Information
 * @rep:category IREP_CLASS_SUBTYPE JAVA_BEAN_SERVICES
 */
public class EmployeeInfo {

  public EmployeeInfo() {
    super();
  }

  /**
     * This method returns a list of direct reports of the requesting
user.
     *
     * @return List of person records who are direct reports
     * @rep:paraminfo {@rep:innertype oracle.apps.per.sample.beans.
PersonBean}
     * @rep:scope public
     * @rep:displayname Get Direct Reports
     * @rep:httpverb get
     * @rep:category BUSINESS_ENTITY sample
     */
  // Demonstration of list return type
 public List<PersonBean> getDirectReports() throws PerServiceException {

...

/**
     * This method returns an array of all reports of the requesting
user.
     *
     * @return Array of person records who are reporting into the
requesting user's organization hierarchy
     * @rep:scope public
     * @rep:displayname Get All Reports
     * @rep:httpverb get
     * @rep:category BUSINESS_ENTITY sample
     */
  // Demonstration of array return type
 public PersonBean[] getAllReports() throws PerServiceException {

 ...
  }

  /**
   * This method returns the person details for a specific person id.
Throws error if the person
   * is not in requesting user's org hierarchy.
   *
   * @return Details of a person in the logged on user's org hierarchy.
   * @param personId Person Identifier
   * @rep:paraminfo {@rep:required} {@rep:key_param}
   * @rep:scope public
   * @rep:displayname Get Person Details
   * @rep:httpverb get
```

```
* @rep:category BUSINESS_ENTITY sample
   */
  // Demonstration of simple navigation using path param
  public PersonBean getPersonInfo(int personId) throws
PerServiceException {

...
```

Here is an example of an annotated Purchase Order service:

```
...
package oracle.apps.po.tutorial;

import oracle.jbo.domain.Number;

import oracle.svc.data.DataList;
import oracle.svc.data.DataService;
import oracle.svc.msg.MessageService;

import oracle.apps.fnd.common.VersionInfo;

/**
 * The Purchase Order service lets you to view, update, acknowledge and
 * approve purchase orders. It also lets you receive items, and obtain
 * pricing by line item.
 *
 * @see oracle.apps.fnd.framework.toolbox.tutorial.PurchaseOrderSDO
 * @see oracle.apps.fnd.framework.toolbox.tutorial.
PurchaseOrderAcknowledgementsSDO
 * @see oracle.apps.fnd.framework.toolbox.tutorial.
PurchaseOrderReceiptsSDO
 *
 * @rep:scope public
 * @rep:displayname Purchase Order Service
 * @rep:implementation oracle.apps.fnd.framework.toolbox.tutorial.
server.PurchaseOrderSAMImpl
 * @rep:product PO
 * @rep:category BUSINESS_ENTITY PO_PURCHASE_ORDER
 * @rep:service
 */
public interface PurchaseOrder extends DataService, MessageService
{
  public static final String RCS_ID="$Header$";
  public static final boolean RCS_ID_RECORDED =
        VersionInfo.recordClassVersion(RCS_ID, "oracle.apps.fnd.
framework.toolbox.tutorial");

 /**
  * Approves a purchase order.
  *
  * @param purchaseOrder purchase order unique identifier
  * @rep:paraminfo {@rep:required}
  *
  * @rep:scope public
  * @rep:displayname Approve Purchase Orders
  * @rep:businessevent oracle.apps.po.approve
  */
  public void approvePurchaseOrder(Number poNumber);

 /**
  * Acknowledges purchase orders, including whether the terms have
  * been accepted or not.  You can also provide updated line
  * item pricing and shipment promise dates with the acknowledgement.
  *
  * @param purchaseOrders list of purchase order objects
  * @rep:paraminfo {@rep:innertype oracle.apps.fnd.framework.toolbox.
tutorial.PurchaseOrderAcknowledgementsSDO} {@required}
  *
  * @rep:scope public
  * @rep:displayname Receive Purchase Order Items
  * @rep:businessevent oracle.apps.po.acknowledge
  */
  public void acknowledgePurchaseOrders(DataList purchaseOrders);

 /**
  * Receives purchase order items. For each given purchase order
```

```
* shipment, indicate the quantity to be received and, optionally,
  * the receipt date if today's date is not an acceptable receipt date.
  *
  * @param purchaseOrders list of purchase order objects
  * @rep:paraminfo {@rep:innertype oracle.apps.fnd.framework.toolbox.
tutorial.PurchaseOrderReceiptsSDO} {@required}
  *
  * @rep:scope public
  * @rep:displayname Receive Purchase Order Items
  * @rep:businessevent oracle.apps.po.receive_item
  */
  public void receiveItems(DataList purchaseOrders);

  /**
   * Gets the price for a purchase order line item.
   *
   * @param poNumber purchase order unique identifier
   * @rep:paraminfo {@required}
   * @param lineNumber purchase order line unique identifier
   * @rep:paraminfo {@required}
   * @return the item price for the given purchase order line
   *
   * @rep:scope public
   * @rep:displayname Get Purchase Order Line Item Price
   */
  public Number getItemPrice(Number poNumber,
                             Number lineNumber);
```

Here is an example of an annotated Purchase Order SDO data object:

```
/*=========================================================================
=====+
 |      Copyright (c) 2004 Oracle Corporation, Redwood Shores, CA, USA
|
 |                         All rights reserved.
|

+=========================================================================
====+
 |  HISTORY
|

+=========================================================================
====*/
package oracle.apps.po.tutorial;

import oracle.jbo.domain.Number;

import oracle.svc.data.DataObjectImpl;
import oracle.svc.data.DataList;

/**
 * The Purchase Order Data Object holds the purchase order data
including
 * nested data objects such as lines and shipments.
 *
 * @see oracle.apps.fnd.framework.toolbox.tutorial.PurchaseOrderLineSDO
 *
 * @rep:scope public
 * @rep:displayname Purchase Order Data Object
 * @rep:product PO
 * @rep:category BUSINESS_ENTITY PO_PURCHASE_ORDER
 * @rep:servicedoc
 */
public class PurchaseOrderSDO extends DataObjectImpl
{
  public PurchaseOrderSDO ()
  {
    super();
  }

  /**
   * Returns the purchase order header id.
   *
   * @return purchase order header id.
   */
  public Number getHeaderId()
  {
    return (Number)getAttribute("HeaderId");
  }

  /**
   * Sets the purchase order header id.
   *
   * @param value purchase order header id.
   * @rep:paraminfo {@rep:precision 5} {@rep:required}
   */
  public void setHeaderId(Number value)
  {
    setAttribute("HeaderId", value);
  }

  /**
   * Returns the purchase order name.
   *
   * @return purchase order name.
```

```
* @rep:paraminfo {rep:precision 80}
   */
  public String getName()
  {
    return (String)getAttribute("Name");
  }

  /**
   * Sets the purchase order header name.
   *
   * @param value purchase order header name.
   * @rep:paraminfo {@rep:precision 80}
   */
  public void setName(String value)
  {
    setAttribute("Name", value);
  }

  /**
   * Returns the purchase order description.
   *
   * @return purchase order description.
   * @rep:paraminfo {rep:precision 120}
   */
  public String getDescription()
  {
    return (String)getAttribute("Description");
  }

  /**
   * Sets the purchase order header description.
   *
   * @param value purchase order header description.
   * @rep:paraminfo {@rep:precision 80}
   */
  public void setDescription(String value)
  {
    setAttribute("Description", value);
  }

  /**
   * @return the purchase order lines DataList.
   * @rep:paraminfo {@rep:innertype oracle.apps.fnd.framework.toolbox.
tutorial.PurchaseOrderLineSDO}
   */
  public DataList getLines()
  {
    return (DataList)getAttribute("Lines");
  }

  /**
   * @param list the putrchase order lines DataList.
   * @rep:paraminfo {@rep:innertype oracle.apps.fnd.framework.toolbox.
tutorial.PurchaseOrderLineSDO}
   */
  public void setLines(DataList list)
  {
    setAttribute("Lines", list);
  }

}
```

PL/SQL Annotations

You can annotate *.pls and *.pkh files.

For PL/SQL packages, only the package spec should be annotated. Do not annotate the body.

Before annotating, make sure that no comments beginning with `/*#` are present. The "slash-star-pound" characters are used to set off repository annotations, and will result in either an error or undesirable behavior if used with normal comments.

To annotate, use a text editor (such as emacs or vi.) to edit the file. For each package, begin your annotations at the second line immediately after the `CREATE OR REPLACE PACKAGE <package_name> AS` line. (The first line after `CREATE OR REPLACE PACKAGE <package_name> AS` should be the `/* $Header: $ */ line`.)

Required Class-level Annotations

- must begin with description sentence(s), page A-116
- rep:scope, page A-119
- rep:product, page A-120
- rep:displayname, page A-121
- rep:category, page A-128

  Use BUSINESS_ENTITY at the class level only if all underlying methods have the same business entity. In those cases, you do not need to repeat the annotation at the method level.

- rep:businessevent, page A-137 (if an event is raised)

Optional Class-level Annotations

- link, page A-125
- see, page A-126
- rep:lifecycle, page A-123
- rep:compatibility, page A-124
- rep:ihelp, page A-127
- rep:metalink, page A-128

Required Method-level Annotations

- must begin with description sentence(s), page A-116
- param, page A-134

  Use only when applicable and when other tags such as `@see` and `@rep:metalink` do not provide parameter explanations.
- return, page A-135 (if applicable)
- rep:displayname, page A-121
- rep:paraminfo, page A-135
- rep:businessevent, page A-137 (if an event is raised)

Optional Method-level Annotations

- link, page A-125
- see, page A-126
- rep:scope, page A-119
- rep:lifecycle, page A-123
- rep:compatibility, page A-124
- rep:category, page A-128

  Use BUSINESS_ENTITY at the method level only when a class methods have heterogeneous business entities.
- rep:ihelp, page A-127
- rep:metalink, page A-128
- rep:appscontext, page A-141
- rep:primaryinstance, page A-142

Template

You can use the following template when annotating PL/SQL files:

```
.
.
.
CREATE OR REPLACE PACKAGE <package name> AS
/* $Header: $ */
/*#
  * <Put your long package description here
  * it can span multiple lines>
  * @rep:scope <scope>
  * @rep:product <product or pseudoproduct short code>
  * @rep:lifecycle <lifecycle>
  * @rep:displayname <display name>
  * @rep:compatibility <compatibility code>
  * @rep:businessevent <Business event name>
  * @rep:category BUSINESS_ENTITY <entity name>
  */


 .
 .
 .


/**
 * <Put your long procedure description here
 * it can span multiple lines>
 * @param <param name 1> <param description 1>
 * @param <param name 2> <param description 2>
 * @rep:scope <scope>
 * @rep:product <product or pseudoproduct short code>
 * @rep:lifecycle <lifecycle>
 * @rep:displayname <display name>
 * @rep:compatibility <compatibility code>
 * @rep:businessevent <Business event name>
 */
PROCEDURE <procedure name> ( . . .);

.
.
.

/**
 * <Put your long function description here
 * it can span multiple lines>
 * @param <param name 1> <param description 1>
 * @param <param name 2> <param description 2>
 * @return <return description>
 * @rep:scope <scope>
 * @rep:product <product or pseudoproduct short code>
 * @rep:lifecycle <lifecycle>
 * @rep:displayname <display name>
 * @rep:compatibility <compatibility code>
 * @rep:businessevent <Business event name>
 */
FUNCTION <function name> ( . . .);

.
.
.

END <package name>;
/

commit;
exit;
```

Example

For reference, here is an example of an annotated PL/SQL file:

```
set verify off
whenever sqlerror exit failure rollback;
whenever oserror exit failure rollback;


create or replace package WF_ENGINE as

/*#
 * This is the public interface for the Workflow engine.  It allows
 * execution of various WF engine functions.
 * @rep:scope public
 * @rep:product WF
 * @rep:displayname Workflow Engine
 * @rep:lifecycle active
 * @rep:compatibility S
 * @rep:category BUSINESS_ENTITY WF_WORKFLOW_ENGINE
 */


g_nid number;          -- current notification id
g_text varchar2(2000); -- text information



--
-- AddItemAttr (PUBLIC)
--   Add a new unvalidated run-time item attribute.
-- IN:
--   itemtype - item type
--   itemkey - item key
--   aname - attribute name
--   text_value   - add text value to it if provided.
--   number_value - add number value to it if provided.
--   date_value   - add date value to it if provided.
-- NOTE:
--   The new attribute has no type associated.  Get/set usages of the
--   attribute must insure type consistency.
--
/*#
 * Adds Item Attribute
 * @param itemtype item type
 * @param itemkey item key
 * @param aname attribute name
 * @param text_value add text value to it if provided.
 * @param number_value add number value to it if provided.
 * @param date_value add date value to it if provided.
 * @rep:scope public
 * @rep:lifecycle active
 * @rep:displayname Add Item Attribute
 */
procedure AddItemAttr(itemtype in varchar2,
                      itemkey in varchar2,
                      aname in varchar2,
                      text_value   in varchar2 default null,
                      number_value in number   default null,
                      date_value   in date     default null);

--
-- AddItemAttrTextArray (PUBLIC)
--   Add an array of new unvalidated run-time item attributes of type
text.
-- IN:
--   itemtype - item type
--   itemkey - item key
--   aname - Array of Names
--   avalue - Array of New values for attribute
```

```
-- NOTE:
--   The new attributes have no type associated.  Get/set usages of
these
--   attributes must insure type consistency.
--


END WF_ENGINE;
/

commit;
exit;
```

Concurrent Program Annotations

To annotate a concurrent program, select the System Administration responsibility and click OA Framework based Define Concurrent Program page. Query the concurrent program and go to the Annotations field. Enter your annotations there and commit to save your work.

After annotating the concurrent program and committing the change, you will need to use FNDLOAD to re-create the LDTs for your concurrent programs.

```
$FND_TOP/bin/FNDLOAD <db_connect> 0 Y DOWNLOAD
$FND_TOP/patch/115/import/afcpprog.lct <ldt_file_name>.ldt PROGRAM
APPLICATION_SHORT_NAME="<application_short_name>"
CONCURRENT_PROGRAM_NAME="<cp_short_name>"
```

Required Class-level Annotations

- must begin with description sentence(s), page A-116

  The annotation takes precedence over the concurrent program own definition in the LDT. One or the other must exist; otherwise, interface generation will fail.

- rep:scope, page A-119

- rep:product, page A-120

- rep:displayname, page A-121

  The annotation takes precedence over the concurrent program own definition in the LDT. One or the other must exist; otherwise, interface generation will fail.

- rep:category, page A-128

- rep:businessevent, page A-137 (if an event is raised)

> **Note:** There is no required method-level annotations for concurrent programs.

Optional Class-level Annotations

- link, page A-125
- see, page A-126
- rep:lifecycle, page A-123
- rep:compatibility, page A-124
- rep:ihelp, page A-127
- rep:metalink, page A-128
- rep:usestable, page A-130
- rep:usesmap, page A-142

> **Note:** There is no optional method-level annotations for concurrent programs.

Template

You can use the following template when annotating Concurrent Programs:

```
/*#
 * <Put your long description here
 * it can span multiple lines>
 * @rep:scope <scope>
 * @rep:product <product or pseudoproduct short code>
 * @rep:lifecycle <lifecycle>
 * @rep:category OPEN_INTERFACE <open interface name> <sequence_num>
 * @rep:usestable <table or view name> <sequence_num> <direction>
 * @rep:category BUSINESS_ENTITY <BO type>
 * @rep:category  <other category> <other value>
 * @rep:businessevent <name of business event>
 */
```

Example

For reference, here is an example of an annotated Concurrent Program:

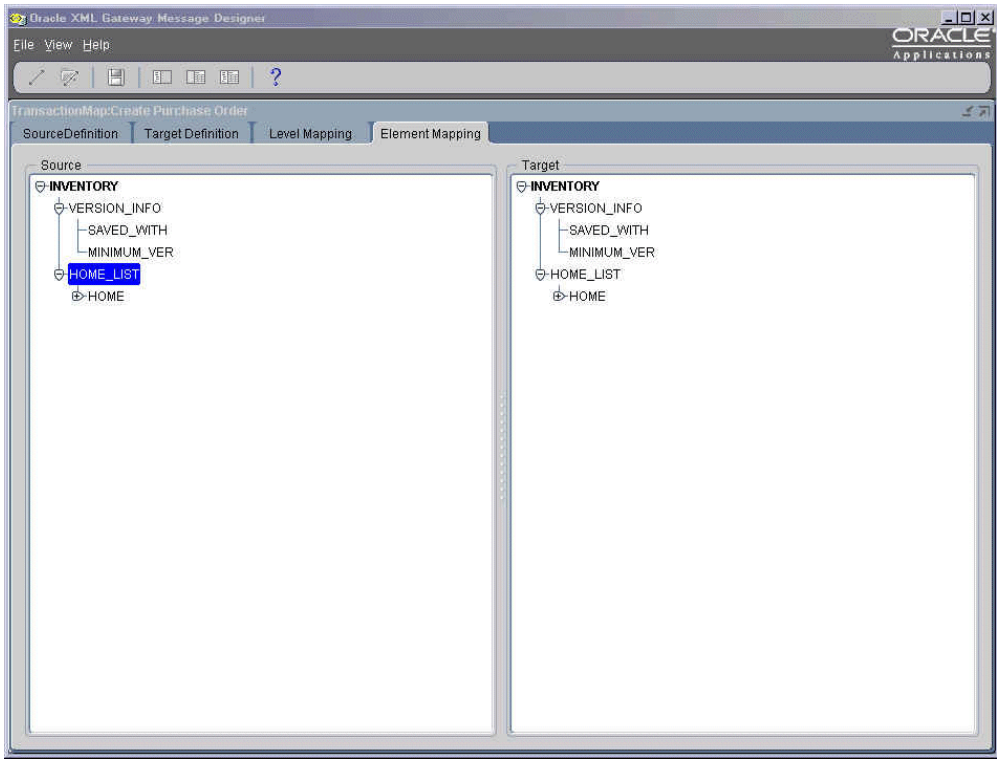
```
/**
 * Executes the Open Interface for Accounts Payable Invoices.  It uses
the
 * following tables: AP_INVOICES_INTERFACE, AP_INVOICE_LINES_INTERFACE.
 * @rep:scope public
 * @rep:product AP
 * @rep:lifecycle active
 * @rep:category OPEN_INTERFACES AP_INVOICES_INTERFACE 1
 * @rep:usestable AP_INVOICES_INTERFACE 2 IN
 * @rep:usestable AP_INVOICE_LINES_INTERFACE 3 IN
 * @rep:category BUSINESS_ENTITY AP_INVOICE
 */
```

XML Gateway Annotations

Use the following procedure to annotate an XML Gateway map for transaction information:

1. Check out an existing map from source code and open it in Message Designer.

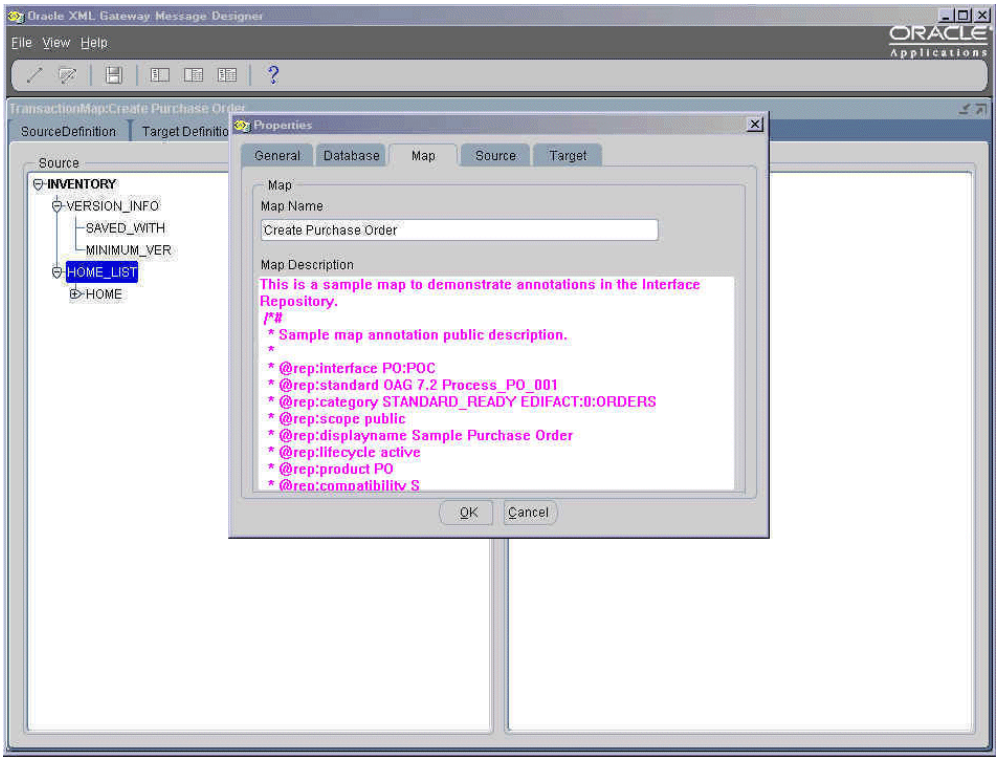
Message Designer Window: Element Mapping Tab



2. Find out which Internal Transaction Type, Subtype, Standard, and Direction this particular map is associated with. Note that this entry must exist in XML Gateway to be loaded into the Integration Repository.

   Click **Message Designer File.... > Properties** and select the Map tab. Annotate the map using the Map Description field after your existing description. Be sure to enter the `@rep:interface` annotation with `<Internal Transaction Type>:<Subtype>`, `@rep:standard`, and `@rep:direction` accordingly.

Properties Dialog: Map Tab



(Optional) If this map is designed to fully support a given standard such as OAG, then set `@rep:standard` to the standard, version and spec name. However, if the map is designed with the intention of supporting standards through additional custom transformations (such as, it is "ready" for the standard), then use the rep: category_STANDARD_READY, page A-128 annotation to denote this.

- A given Internal Transaction Type and Subtype should have only one map seeded by product teams for a given Standard and Direction (regardless of Party Type). Additional maps containing the same types in the annotations would be rejected and treated as errors. Note that there may exist different maps based on the External Transaction Type and Subtype, but as these are meant to be Trading Partner-specific, we do not enter them in the repository. In future releases, we will enforce these rules natively within XML Gateway.

- If a single map is reused in more than one Internal Transaction Type and Subtype, then you may enter multiple annotations, each within its own comment block (i.e. between /*# ... */). The parser will create entries in the Integration Repository for each

annotation set. Although this capability is supported, you are encouraged to use two different maps to accommodate potentially changing interfaces. See the following example of map reuse:

Int T	Int ST	D	Ext T	Ext ST	STD	Party Type
AR	Invoice	O	Invoice	Process	OAG	C
AR	Credit	O	Invoice	Process	OAG	C
AR	Debit	O	Invoice	Process	OAG	C

In this scenario, since the external representation does not change, the same map can be reused. However, the internal processing and authorization considerations may differ based on the Internal Transaction Type and Subtype. In this case, the map can have three annotation blocks, one for each Internal Transaction Type and Subtype; such as. AR-Invoice, AR-Credit, and AR-Debit.

- Parameters are typically used in outbound maps for specifying keys used in queries to produce outbound data. Inbound maps do not have parameters.

3. Save the annotated map, check it into source control, and release as a patch as usual. The annotations are updated as part of the Integration Repository loaders.

Required Class-level Annotations

- must begin with description sentence(s), page A-116
- rep:scope, page A-119
- rep:product, page A-120
- rep:displayname, page A-121
- rep:category, page A-128
- rep:standard, page A-131

- rep:interface, page A-133
- rep:businessevent, page A-137 (if an event is raised)
- rep:direction, page A-137

Note: There is no required method-level annotations for XML Gateway.

Optional Class-level Annotations

- link, page A-125
- see, page A-126
- param, page A-134

  Use only when applicable and when other tags such as `@see` and `@rep:metalink` do not provide parameter explanations.
- rep:paraminfo, page A-135
- rep:lifecycle, page A-123
- rep:compatibility, page A-124
- rep:ihelp, page A-127
- rep:metalink, page A-128
- rep:synchronicity, page A-140

Note: There is no optional method-level annotations for XML Gateway.

Template

You can use the following template when annotating XML Gateway:

```
Sample Inbound Map Annotation

 /*#
  * Sample map annotation public description.
  *
  * @rep:interface <transaction_type:sub_type>
  * @rep:standard <OAG|cXML> <7.2|7.3> <specname>
  * @rep:direction IN
  * @rep:scope <public|private|internal>
  * @rep:displayname <Interface display name>
  * @rep:lifecycle <active|deprecated|obsolete|planned>
  * @rep:product <product code>
  * @rep:compatibility <S|N>
  * @rep:category <lookupType> <lookupCode> <sequenceNumber>
  * @rep:category STANDARD_READY <standard:version:specification>
  * @rep:businessevent <businessEventName>
  */

Sample Outbound Map Annotation

 /*#
  * Sample map annotation public description.
  *
  * @param <paramName> <Parameter description>
  * @rep:paraminfo {@rep:required}
  *
  * @rep:interface <transaction_type:sub_type>
  * @rep:standard <OAG|cXML> <7.2|7.3> <specname>
  * @rep:direction OUT
  * @rep:scope <public|private|internal>
  * @rep:displayname <Interface display name>
  * @rep:lifecycle <active|deprecated|obsolete|planned>
  * @rep:product <product code>
  * @rep:compatibility <S|N>
  * @rep:category <lookupType> <lookupCode> <sequenceNumber>
  * @rep:category STANDARD_READY <standard:version:specification>
  * @rep:businessevent <businessEventName>
  */

Important Note

A given map should be unique to a given Internal Transaction Type /
Subtype, Standard and Direction. This is because the External
Transaction Type / Subtype are meant for Trading Partner specific values
to be specified in the Trading Partner Details form and the entries in
the Integration Repository are NOT Trading Partner specific. Moreover,
there should not be a need to change maps on a per Trading Partner
basis, and if it does, then those maps should not be part of the
Integration Repository entries.
Given the current data model however, it is possible that a given map
could differ by External Transaction Type / Subtype and even by Trading
Partner. Going forward, this would not be allowed for seeded maps and
the Integration Repository parser would return an error if it finds
multiple maps which point to the same Internal Transaction Type /
Subtype.
Additional Notes

    * Parameters are typically used in outbound maps for specifying keys
used in queries to produce outbound data
```

Example

For reference, here is an example of an annotated XML Gateway interface:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- $Header: MapPrinter.java 115.12 2009/06/13 21:17:58 mtai noship $
-->
<!-- WARNING: This file should only be edited using Message Designer -->
<?xGateway mapType="MAP" ?>
<?xGatewayVersion designerVersion="2.6.3.0.0" ?>
<ECX_MAPPINGS>
<MAP_CODE>Create Purchase Order</MAP_CODE>
<DESCRIPTION>This is a sample map to demonstrate annotations in the
Interface Repository.
 /*#
  * Sample map annotation public description.
  *
  * @rep:interface PO:POC
  * @rep:standard OAG 7.2 Process_PO_001
  * @rep:direction IN
  * @rep:scope public
  * @rep:displayname Create Purchase Order
  * @rep:lifecycle active
  * @rep:product PO
  * @rep:compatibility S
  * @rep:category BUSINESS_OBJECT PURCHASE_ORDER
  * @rep:businessevent oracle.apps.po.received
  */
 /*#
  * Sample map annotation public description for reused transaction
  *
  * @rep:interface PO:POU
  * @rep:standard OAG 7.2 Process_PO_001
  * @rep:direction IN
  * @rep:scope public
  * @rep:displayname Update Purchase Order
  * @rep:lifecycle active
  * @rep:product PO
  * @rep:compatibility S
  * @rep:category BUSINESS_OBJECT PURCHASE_ORDER
  * @rep:businessevent oracle.apps.po.received
  */
</DESCRIPTION>
<OBJECT_ID_SOURCE>1</OBJECT_ID_SOURCE>
<OBJECT_ID_TARGET>2</OBJECT_ID_TARGET>
<ENABLED>Y</ENABLED>
<ECX_MAJOR_VERSION>2</ECX_MAJOR_VERSION>
<ECX_MINOR_VERSION>6</ECX_MINOR_VERSION>
<ECX_OBJECTS>
<OBJECT_ID>1</OBJECT_ID>
<OBJECT_NAME>SRC</OBJECT_NAME>
<OBJECT_TYPE>XML</OBJECT_TYPE>
<OBJECT_DESCRIPTION>Source Definition</OBJECT_DESCRIPTION>
<OBJECT_STANDARD>OAG</OBJECT_STANDARD>
<ROOT_ELEMENT>INVENTORY</ROOT_ELEMENT>


<ECX_OBJECT_LEVELS>
<OBJECTLEVEL_ID>0</OBJECTLEVEL_ID>
<OBJECT_ID>1</OBJECT_ID>
<OBJECT_LEVEL>0</OBJECT_LEVEL>
<OBJECT_LEVEL_NAME>INVENTORY</OBJECT_LEVEL_NAME>
<PARENT_LEVEL>0</PARENT_LEVEL>
<ENABLED>Y</ENABLED>
<ECX_OBJECT_ATTRIBUTES>
<OBJECTLEVEL_ID>0</OBJECTLEVEL_ID>
<ATTRIBUTE_ID>0</ATTRIBUTE_ID>
<ATTRIBUTE_NAME>INVENTORY</ATTRIBUTE_NAME>
<OBJECT_COLUMN_FLAG>N</OBJECT_COLUMN_FLAG>
```

```
<PARENT_ATTRIBUTE_ID>0</PARENT_ATTRIBUTE_ID>


<ATTRIBUTE_TYPE>1</ATTRIBUTE_TYPE>
<HAS_ATTRIBUTES>0</HAS_ATTRIBUTES>
<LEAF_NODE>1</LEAF_NODE>
<REQUIRED_FLAG>Y</REQUIRED_FLAG>
<IS_MAPPED>false</IS_MAPPED>
</ECX_OBJECT_ATTRIBUTES>
<ECX_OBJECT_ATTRIBUTES>
<OBJECTLEVEL_ID>0</OBJECTLEVEL_ID>
<ATTRIBUTE_ID>1</ATTRIBUTE_ID>
<ATTRIBUTE_NAME>VERSION_INFO</ATTRIBUTE_NAME>
<OBJECT_COLUMN_FLAG>N</OBJECT_COLUMN_FLAG>
<DATA_TYPE>VARCHAR2</DATA_TYPE>
<PARENT_ATTRIBUTE_ID>0</PARENT_ATTRIBUTE_ID>

<ATTRIBUTE_TYPE>1</ATTRIBUTE_TYPE>
<HAS_ATTRIBUTES>0</HAS_ATTRIBUTES>
<LEAF_NODE>1</LEAF_NODE>
<REQUIRED_FLAG>N</REQUIRED_FLAG>
<IS_MAPPED>false</IS_MAPPED>
</ECX_OBJECT_ATTRIBUTES>
<ECX_OBJECT_ATTRIBUTES>
<OBJECTLEVEL_ID>0</OBJECTLEVEL_ID>
<ATTRIBUTE_ID>2</ATTRIBUTE_ID>
<ATTRIBUTE_NAME>SAVED_WITH</ATTRIBUTE_NAME>
<OBJECT_COLUMN_FLAG>N</OBJECT_COLUMN_FLAG>
<DATA_TYPE>VARCHAR2</DATA_TYPE>
<PARENT_ATTRIBUTE_ID>1</PARENT_ATTRIBUTE_ID>

<ATTRIBUTE_TYPE>1</ATTRIBUTE_TYPE>
<HAS_ATTRIBUTES>0</HAS_ATTRIBUTES>
<LEAF_NODE>1</LEAF_NODE>
<REQUIRED_FLAG>N</REQUIRED_FLAG>
<IS_MAPPED>false</IS_MAPPED>
</ECX_OBJECT_ATTRIBUTES>
<ECX_OBJECT_ATTRIBUTES>
<OBJECTLEVEL_ID>0</OBJECTLEVEL_ID>
<ATTRIBUTE_ID>3</ATTRIBUTE_ID>
<ATTRIBUTE_NAME>MINIMUM_VER</ATTRIBUTE_NAME>
<OBJECT_COLUMN_FLAG>N</OBJECT_COLUMN_FLAG>
<DATA_TYPE>VARCHAR2</DATA_TYPE>
<PARENT_ATTRIBUTE_ID>1</PARENT_ATTRIBUTE_ID>

<ATTRIBUTE_TYPE>1</ATTRIBUTE_TYPE>
<HAS_ATTRIBUTES>0</HAS_ATTRIBUTES>
<LEAF_NODE>1</LEAF_NODE>
<REQUIRED_FLAG>N</REQUIRED_FLAG>
<IS_MAPPED>false</IS_MAPPED>
</ECX_OBJECT_ATTRIBUTES>
<ECX_OBJECT_ATTRIBUTES>
<OBJECTLEVEL_ID>0</OBJECTLEVEL_ID>
<ATTRIBUTE_ID>4</ATTRIBUTE_ID>
<ATTRIBUTE_NAME>HOME_LIST</ATTRIBUTE_NAME>
<OBJECT_COLUMN_FLAG>N</OBJECT_COLUMN_FLAG>
<DATA_TYPE>VARCHAR2</DATA_TYPE>
<PARENT_ATTRIBUTE_ID>0</PARENT_ATTRIBUTE_ID>

<ATTRIBUTE_TYPE>1</ATTRIBUTE_TYPE>
<HAS_ATTRIBUTES>0</HAS_ATTRIBUTES>
<LEAF_NODE>1</LEAF_NODE>
<REQUIRED_FLAG>N</REQUIRED_FLAG>
<IS_MAPPED>false</IS_MAPPED>
</ECX_OBJECT_ATTRIBUTES>
<ECX_OBJECT_ATTRIBUTES>
```

```
<OBJECTLEVEL_ID>0</OBJECTLEVEL_ID>
<ATTRIBUTE_ID>5</ATTRIBUTE_ID>
<ATTRIBUTE_NAME>HOME</ATTRIBUTE_NAME>
<OBJECT_COLUMN_FLAG>N</OBJECT_COLUMN_FLAG>
<DATA_TYPE>VARCHAR2</DATA_TYPE>
<PARENT_ATTRIBUTE_ID>4</PARENT_ATTRIBUTE_ID>

<ATTRIBUTE_TYPE>1</ATTRIBUTE_TYPE>
<HAS_ATTRIBUTES>4</HAS_ATTRIBUTES>
<LEAF_NODE>0</LEAF_NODE>
<REQUIRED_FLAG>N</REQUIRED_FLAG>
<IS_MAPPED>false</IS_MAPPED>
</ECX_OBJECT_ATTRIBUTES>
<ECX_OBJECT_ATTRIBUTES>
<OBJECTLEVEL_ID>0</OBJECTLEVEL_ID>
<ATTRIBUTE_ID>6</ATTRIBUTE_ID>
<ATTRIBUTE_NAME>NAME</ATTRIBUTE_NAME>
<OBJECT_COLUMN_FLAG>N</OBJECT_COLUMN_FLAG>
<DATA_TYPE>VARCHAR2</DATA_TYPE>
<PARENT_ATTRIBUTE_ID>5</PARENT_ATTRIBUTE_ID>

<ATTRIBUTE_TYPE>2</ATTRIBUTE_TYPE>
<HAS_ATTRIBUTES>0</HAS_ATTRIBUTES>
<LEAF_NODE>1</LEAF_NODE>
<REQUIRED_FLAG>N</REQUIRED_FLAG>
<IS_MAPPED>false</IS_MAPPED>
</ECX_OBJECT_ATTRIBUTES>
<ECX_OBJECT_ATTRIBUTES>
<OBJECTLEVEL_ID>0</OBJECTLEVEL_ID>
<ATTRIBUTE_ID>7</ATTRIBUTE_ID>
<ATTRIBUTE_NAME>LOC</ATTRIBUTE_NAME>
<OBJECT_COLUMN_FLAG>N</OBJECT_COLUMN_FLAG>
<DATA_TYPE>VARCHAR2</DATA_TYPE>
<PARENT_ATTRIBUTE_ID>5</PARENT_ATTRIBUTE_ID>

<ATTRIBUTE_TYPE>2</ATTRIBUTE_TYPE>
<HAS_ATTRIBUTES>0</HAS_ATTRIBUTES>
<LEAF_NODE>1</LEAF_NODE>
<REQUIRED_FLAG>N</REQUIRED_FLAG>
<IS_MAPPED>false</IS_MAPPED>
</ECX_OBJECT_ATTRIBUTES>
<ECX_OBJECT_ATTRIBUTES>
<OBJECTLEVEL_ID>0</OBJECTLEVEL_ID>
<ATTRIBUTE_ID>8</ATTRIBUTE_ID>
<ATTRIBUTE_NAME>TYPE</ATTRIBUTE_NAME>
<OBJECT_COLUMN_FLAG>N</OBJECT_COLUMN_FLAG>
<DATA_TYPE>VARCHAR2</DATA_TYPE>
<PARENT_ATTRIBUTE_ID>5</PARENT_ATTRIBUTE_ID>

<ATTRIBUTE_TYPE>2</ATTRIBUTE_TYPE>
<HAS_ATTRIBUTES>0</HAS_ATTRIBUTES>
<LEAF_NODE>1</LEAF_NODE>
<REQUIRED_FLAG>N</REQUIRED_FLAG>
<IS_MAPPED>false</IS_MAPPED>
</ECX_OBJECT_ATTRIBUTES>
<ECX_OBJECT_ATTRIBUTES>
<OBJECTLEVEL_ID>0</OBJECTLEVEL_ID>
<ATTRIBUTE_ID>9</ATTRIBUTE_ID>
<ATTRIBUTE_NAME>IDX</ATTRIBUTE_NAME>
<OBJECT_COLUMN_FLAG>N</OBJECT_COLUMN_FLAG>
<DATA_TYPE>VARCHAR2</DATA_TYPE>
<PARENT_ATTRIBUTE_ID>5</PARENT_ATTRIBUTE_ID>

<ATTRIBUTE_TYPE>2</ATTRIBUTE_TYPE>
<HAS_ATTRIBUTES>0</HAS_ATTRIBUTES>
<LEAF_NODE>1</LEAF_NODE>
```

```
<REQUIRED_FLAG>N</REQUIRED_FLAG>
<IS_MAPPED>false</IS_MAPPED>
</ECX_OBJECT_ATTRIBUTES>
</ECX_OBJECT_LEVELS>
</ECX_OBJECTS>
<ECX_OBJECTS>
<OBJECT_ID>2</OBJECT_ID>
<OBJECT_NAME>TGT</OBJECT_NAME>
<OBJECT_TYPE>XML</OBJECT_TYPE>
<OBJECT_DESCRIPTION>Target Definition</OBJECT_DESCRIPTION>
<OBJECT_STANDARD>OAG</OBJECT_STANDARD>
<ROOT_ELEMENT>INVENTORY</ROOT_ELEMENT>


<ECX_OBJECT_LEVELS>
<OBJECTLEVEL_ID>1</OBJECTLEVEL_ID>
<OBJECT_ID>2</OBJECT_ID>
<OBJECT_LEVEL>0</OBJECT_LEVEL>
<OBJECT_LEVEL_NAME>INVENTORY</OBJECT_LEVEL_NAME>
<PARENT_LEVEL>0</PARENT_LEVEL>
<ENABLED>Y</ENABLED>
<ECX_OBJECT_ATTRIBUTES>
<OBJECTLEVEL_ID>1</OBJECTLEVEL_ID>
<ATTRIBUTE_ID>0</ATTRIBUTE_ID>
<ATTRIBUTE_NAME>INVENTORY</ATTRIBUTE_NAME>
<OBJECT_COLUMN_FLAG>N</OBJECT_COLUMN_FLAG`

<PARENT_ATTRIBUTE_ID>0</PARENT_ATTRIBUTE_ID>


<ATTRIBUTE_TYPE>1</ATTRIBUTE_TYPE>
<HAS_ATTRIBUTES>0</HAS_ATTRIBUTES>
<LEAF_NODE>1</LEAF_NODE>
<REQUIRED_FLAG>Y</REQUIRED_FLAG>
<IS_MAPPED>false</IS_MAPPED>
</ECX_OBJECT_ATTRIBUTES>
<ECX_OBJECT_ATTRIBUTES>
<OBJECTLEVEL_ID>1</OBJECTLEVEL_ID>
<ATTRIBUTE_ID>1</ATTRIBUTE_ID>
<ATTRIBUTE_NAME>VERSION_INFO</ATTRIBUTE_NAME>
<OBJECT_COLUMN_FLAG>N</OBJECT_COLUMN_FLAG>
<DATA_TYPE>VARCHAR2</DATA_TYPE>
<PARENT_ATTRIBUTE_ID>0</PARENT_ATTRIBUTE_ID>

<ATTRIBUTE_TYPE>1</ATTRIBUTE_TYPE>
<HAS_ATTRIBUTES>0</HAS_ATTRIBUTES>
<LEAF_NODE>1</LEAF_NODE>
<REQUIRED_FLAG>N</REQUIRED_FLAG>
<IS_MAPPED>false</IS_MAPPED>
</ECX_OBJECT_ATTRIBUTES>
<ECX_OBJECT_ATTRIBUTES>
<OBJECTLEVEL_ID>1</OBJECTLEVEL_ID>
<ATTRIBUTE_ID>2</ATTRIBUTE_ID>
<ATTRIBUTE_NAME>SAVED_WITH</ATTRIBUTE_NAME>
<OBJECT_COLUMN_FLAG>N</OBJECT_COLUMN_FLAG>
<DATA_TYPE>VARCHAR2</DATA_TYPE>
<PARENT_ATTRIBUTE_ID>1</PARENT_ATTRIBUTE_ID>

<ATTRIBUTE_TYPE>1</ATTRIBUTE_TYPE>
<HAS_ATTRIBUTES>0</HAS_ATTRIBUTES>
<LEAF_NODE>1</LEAF_NODE>
<REQUIRED_FLAG>N</REQUIRED_FLAG>
<IS_MAPPED>false</IS_MAPPED>
</ECX_OBJECT_ATTRIBUTES>
<ECX_OBJECT_ATTRIBUTES>
<OBJECTLEVEL_ID>1</OBJECTLEVEL_ID>
```

```
<ATTRIBUTE_ID>3</ATTRIBUTE_ID>
<ATTRIBUTE_NAME>MINIMUM_VER</ATTRIBUTE_NAME>
<OBJECT_COLUMN_FLAG>N</OBJECT_COLUMN_FLAG>
<DATA_TYPE>VARCHAR2</DATA_TYPE>
<PARENT_ATTRIBUTE_ID>1</PARENT_ATTRIBUTE_ID>

<ATTRIBUTE_TYPE>1</ATTRIBUTE_TYPE>
<HAS_ATTRIBUTES>0</HAS_ATTRIBUTES>
<LEAF_NODE>1</LEAF_NODE>
<REQUIRED_FLAG>N</REQUIRED_FLAG>
<IS_MAPPED>false</IS_MAPPED>
</ECX_OBJECT_ATTRIBUTES>
<ECX_OBJECT_ATTRIBUTES>
<OBJECTLEVEL_ID>1</OBJECTLEVEL_ID>
<ATTRIBUTE_ID>4</ATTRIBUTE_ID>
<ATTRIBUTE_NAME>HOME_LIST</ATTRIBUTE_NAME>
<OBJECT_COLUMN_FLAG>N</OBJECT_COLUMN_FLAG>
<DATA_TYPE>VARCHAR2</DATA_TYPE>
<PARENT_ATTRIBUTE_ID>0</PARENT_ATTRIBUTE_ID>

<ATTRIBUTE_TYPE>1</ATTRIBUTE_TYPE>
<HAS_ATTRIBUTES>0</HAS_ATTRIBUTES>
<LEAF_NODE>1</LEAF_NODE>
<REQUIRED_FLAG>N</REQUIRED_FLAG>
<IS_MAPPED>false</IS_MAPPED>
</ECX_OBJECT_ATTRIBUTES>
<ECX_OBJECT_ATTRIBUTES>
<OBJECTLEVEL_ID>1</OBJECTLEVEL_ID>
<ATTRIBUTE_ID>5</ATTRIBUTE_ID>
<ATTRIBUTE_NAME>HOME</ATTRIBUTE_NAME>
<OBJECT_COLUMN_FLAG>N</OBJECT_COLUMN_FLAG>
<DATA_TYPE>VARCHAR2</DATA_TYPE>
<PARENT_ATTRIBUTE_ID>4</PARENT_ATTRIBUTE_ID>

<ATTRIBUTE_TYPE>1</ATTRIBUTE_TYPE>
<HAS_ATTRIBUTES>4</HAS_ATTRIBUTES>
<LEAF_NODE>0</LEAF_NODE>
<REQUIRED_FLAG>N</REQUIRED_FLAG>
<IS_MAPPED>false</IS_MAPPED>
</ECX_OBJECT_ATTRIBUTES>
<ECX_OBJECT_ATTRIBUTES>
<OBJECTLEVEL_ID>1</OBJECTLEVEL_ID>
<ATTRIBUTE_ID>6</ATTRIBUTE_ID>
<ATTRIBUTE_NAME>NAME</ATTRIBUTE_NAME>
<OBJECT_COLUMN_FLAG>N</OBJECT_COLUMN_FLAG>
<DATA_TYPE>VARCHAR2</DATA_TYPE>
<PARENT_ATTRIBUTE_ID>5</PARENT_ATTRIBUTE_ID>

<ATTRIBUTE_TYPE>2</ATTRIBUTE_TYPE>
<HAS_ATTRIBUTES>0</HAS_ATTRIBUTES>
<LEAF_NODE>1</LEAF_NODE>
<REQUIRED_FLAG>N</REQUIRED_FLAG>
<IS_MAPPED>false</IS_MAPPED>
</ECX_OBJECT_ATTRIBUTES>
<ECX_OBJECT_ATTRIBUTES>
<OBJECTLEVEL_ID>1</OBJECTLEVEL_ID>
<ATTRIBUTE_ID>7</ATTRIBUTE_ID>
<ATTRIBUTE_NAME>LOC</ATTRIBUTE_NAME>
<OBJECT_COLUMN_FLAG>N</OBJECT_COLUMN_FLAG>
<DATA_TYPE>VARCHAR2</DATA_TYPE>
<PARENT_ATTRIBUTE_ID>5</PARENT_ATTRIBUTE_ID>

<ATTRIBUTE_TYPE>2</ATTRIBUTE_TYPE>
<HAS_ATTRIBUTES>0</HAS_ATTRIBUTES>
<LEAF_NODE>1</LEAF_NODE>
<REQUIRED_FLAG>N</REQUIRED_FLAG>
```

```
<IS_MAPPED>false</IS_MAPPED>
</ECX_OBJECT_ATTRIBUTES>
<ECX_OBJECT_ATTRIBUTES>
<OBJECTLEVEL_ID>1</OBJECTLEVEL_ID>
<ATTRIBUTE_ID>8</ATTRIBUTE_ID>
<ATTRIBUTE_NAME>TYPE</ATTRIBUTE_NAME>
<OBJECT_COLUMN_FLAG>N</OBJECT_COLUMN_FLAG>
<DATA_TYPE>VARCHAR2</DATA_TYPE>
<PARENT_ATTRIBUTE_ID>5</PARENT_ATTRIBUTE_ID>

<ATTRIBUTE_TYPE>2</ATTRIBUTE_TYPE>
<HAS_ATTRIBUTES>0</HAS_ATTRIBUTES>
<LEAF_NODE>1</LEAF_NODE>
<REQUIRED_FLAG>N</REQUIRED_FLAG>
<IS_MAPPED>false</IS_MAPPED>
</ECX_OBJECT_ATTRIBUTES>
<ECX_OBJECT_ATTRIBUTES>
<OBJECTLEVEL_ID>1</OBJECTLEVEL_ID>
<ATTRIBUTE_ID>9</ATTRIBUTE_ID>
<ATTRIBUTE_NAME>IDX</ATTRIBUTE_NAME>
<OBJECT_COLUMN_FLAG>N</OBJECT_COLUMN_FLAG>
<DATA_TYPE>VARCHAR2</DATA_TYPE>
<PARENT_ATTRIBUTE_ID>5</PARENT_ATTRIBUTE_ID>

<ATTRIBUTE_TYPE>2</ATTRIBUTE_TYPE>
<HAS_ATTRIBUTES>0</HAS_ATTRIBUTES>
<LEAF_NODE>1</LEAF_NODE>
<REQUIRED_FLAG>N</REQUIRED_FLAG>
<IS_MAPPED>false</IS_MAPPED>
</ECX_OBJECT_ATTRIBUTES>
</ECX_OBJECT_LEVELS>
</ECX_OBJECTS>
</ECX_MAPPINGS>
<SCRIPT  SRC="/oracle_smp_chronos/oracle_smp_chronos.js"></SCRIPT>
```

Business Event Annotations

This section describes what you should know about Integration Repository annotations for business events, and includes the following topics:

- Annotating Business Events
- Annotations for Business Events - Syntax
- Required Annotations
- Optional Annotations
- Template
- Example

Annotating Business Events

- You should annotate business events in *.wfx files.
- You should annotate only events. Subscriptions need not be annotated; they will

not be available in Integration Repository.

- Before annotating, make sure that no comments beginning with /*# are present. These "slash-star-pound" characters are used to mark the start of repository annotations, and will produce errors or unspecified behavior if used in normal comments.

- To annotate, use a text editor such as emacs or vi to edit the file.

- In the .wfx file, place the annotations within the `<IREP_ANNOTATION>` tag for the business event. Note that the `<IREP_ANNOTATION>` tag is a child node of the `<WF_EVENTS>` tag.

- For .wfx files having multiple business event definitions, each of the business event definitions should be separately annotated. That is, you should place the annotation within an `<IREP_ANNOTATION>` tag for the appropriate business events.

- Enter a meaningful description that covers the condition under which the business event is raised, and the UI action that invokes the business event.

- Define product codes in FND_APPLICATION.

- Use existing business entities for your events. For the list of existing business entities, see Business Entity Annotation Guidelines, page A-41.

- If you decide not to annotate or publish the event after all, you should remove the annotation only, and not the associated tags.

  The presence of either the `<IREP_ANNOTATION/>` tag or `<IREP_ANNOTATION></IREP_ANNOTATION>` tag is an indication to the loader that the business event has been reviewed for annotation and does not need to be published to integration repository. The next time the user downloads these events, the loader will insert empty `<IREP_ANNOTATION>` tags.

- If you remove the entire `<IREP_ANNOTATION>` tag for the business event and then upload it, on a subsequent download the loader will insert a partially filled annotation template for the business event.

Annotations for Business Events - Syntax

The annotations for business events are:

```
<IREP_ANNOTATION>
/*#
* This event is raised after the Purchase Order has been pushed
* to Oracle Order management open interface tables. This event
* will start the workflow OEOI/R_OEOI_ORDER_IMPORT to import the
* order.
* @rep:scope public
* @rep:displayname OM Generic Inbound Event
* @rep:product ONT
* @rep:category BUSINESS_ENTITY ONT_SALES_ORDER
*/
</IREP_ANNOTATION>
```

Refer to General Guidelines for Annotations, page A-1 for details of element definitions.

Required Annotations

Follow the links below to view syntax and usage of each annotation.

- Must begin with description sentence(s)
- rep:displayname, page A-121
- rep:scope, page A-119
- rep:product, page A-120
- rep:category BUSINESS_ENTITY, page A-128

Optional Annotations

- link, page A-125
- see, page A-126
- rep:lifecycle, page A-123
- rep:compatibility, page A-124
- rep:ihelp, page A-127
- rep:metalink, page A-128

Template

You can use this template when annotating .wfx files.

```
.
.
.

<oracle.apps.wf.event.all.sync>
.
.
.
<WF_TABLE_DATA>
   <WF_EVENTS>
      <VERSION>...</VERSION>
      <GUID>....</GUID>
      <NAME>event name</NAME>
      <TYPE>EVENT</TYPE>
      <STATUS>ENABLED</STATUS>
      <GENERATE_FUNCTION/>
      <OWNER_NAME> ... </OWNER_NAME>
      <OWNER_TAG>...</OWNER_TAG>
      <CUSTOMIZATION_LEVEL>...</CUSTOMIZATION_LEVEL>
      <LICENSED_FLAG>..</LICENSED_FLAG>
      <DISPLAY_NAME>...</DISPLAY_NAME>
      <DESCRIPTION> Description for business event </DESCRIPTION>
      <IREP_ANNOTATION>
 /*#
  * Put your long package description here; it can span multiple lines.
  *
  * @rep:scope <scope>
  * @rep:displayname <display name>
  * @rep:product <product or pseudoproduct short code>
  * @rep:category BUSINESS_ENTITY <entity name>
  */
   </IREP_ANNOTATION>
   </WF_EVENTS>
</WF_TABLE_DATA>


.
.
.

<WF_TABLE_DATA>
   <WF_EVENTS>
      <VERSION>...</VERSION>
      <GUID>....</GUID>
      <NAME>event name</NAME>
      <TYPE>EVENT</TYPE>
      <STATUS>ENABLED</STATUS>
      <GENERATE_FUNCTION/>
      <OWNER_NAME> ... </OWNER_NAME>
      <OWNER_TAG>...</OWNER_TAG>
      <CUSTOMIZATION_LEVEL>...</CUSTOMIZATION_LEVEL>
      <LICENSED_FLAG>..</LICENSED_FLAG>
      <DISPLAY_NAME>...</DISPLAY_NAME>
      <DESCRIPTION> Description for business event </DESCRIPTION>
      <IREP_ANNOTATION>
 /*#
  * Put your long package description here; it can span multiple lines.
  *
  * @rep:scope <scope>
  * @rep:displayname <display name>
  * @rep:product <product or pseudoproduct short code>
  * @rep:category BUSINESS_ENTITY <entity name>
  */
   </IREP_ANNOTATION>
   </WF_EVENTS>
</WF_TABLE_DATA>
```

```
.
.
.
</oracle.apps.wf.event.all.sync>
```

Example

For reference, here is an example of an annotated .wfx file:

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <!--  $Header: oeevtname.wfx 120.0 2005/06/01 23:11:59 appldev noship
$ -->
- <!--  dbdrv: exec java oracle/apps/fnd/wf WFXLoad.class java
&phase=daa+38 \ -->
- <!--  dbdrv: checkfile(115.2=120.0):~PROD:~PATH:~FILE \ -->
- <!--  dbdrv: -u &un_apps &pw_apps &jdbc_db_addr &jdbc_protocol US \ --
>
- <!--  dbdrv: &fullpath_~PROD_~PATH_~FILE -->
- <oracle.apps.wf.event.all.sync>
- <ExternalElement>
- <OraTranslatibility>
- <XlatElement Name="WF_EVENTS">
- <XlatID>
  <Key>NAME</Key>
  </XlatID>
  <XlatElement Name="DISPLAY_NAME" MaxLen="80" Expansion="50" />
- <XlatID>
  <Key Type="CONSTANT">DISPLAY_NAME</Key>
  </XlatID>
  <XlatElement Name="DESCRIPTION" MaxLen="2000" Expansion="50" />
- <XlatID>
  <Key Type="CONSTANT">DESCRIPTION</Key>
  </XlatID>
  </XlatElement>
  </OraTranslatibility>
  </ExternalElement>
- <WF_TABLE_DATA>
+ <WF_EVENTS>
  <VERSION>1.0</VERSION>
  <GUID>xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx</GUID>
  <NAME>oracle.apps.ont.oi.po_ack.create</NAME>
  <TYPE>EVENT</TYPE>
  <STATUS>ENABLED</STATUS>
  <GENERATE_FUNCTION />
  <OWNER_NAME>Oracle Order Management</OWNER_NAME>
  <OWNER_TAG>ONT</OWNER_TAG>
  <CUSTOMIZATION_LEVEL>L</CUSTOMIZATION_LEVEL>
  <LICENSED_FLAG>Y</LICENSED_FLAG>
  <DISPLAY_NAME>Event for 3A4 Outbound Acknowledgment</DISPLAY_NAME>
  <DESCRIPTION>Event for 3A4 Outbound Acknowledgment</DESCRIPTION>
  <IREP_ANNOTATION>/*# * This event confirms the buyer of the results of
order import. This event will start the workflow
OEOA/R_OEOA_SEND_ACKNOWLEDGMENT. * * @rep:scope public * @rep:
displayname Event for 3A4 Outbound Acknowledgment * @rep:product ONT *
@rep:category BUSINESS_ENTITY ONT_SALES_ORDER */</IREP_ANNOTATION>
  </WF_EVENTS>
  </WF_TABLE_DATA>
- <WF_TABLE_DATA>
- <WF_EVENTS>
  <VERSION>1.0</VERSION>
  <GUID>xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx</GUID>
  <NAME>oracle.apps.ont.oi.po_inbound.create</NAME>
  <TYPE>EVENT</TYPE>
  <STATUS>ENABLED</STATUS>
  <GENERATE_FUNCTION />
  <OWNER_NAME>Oracle Order Management</OWNER_NAME>
  <OWNER_TAG>ONT</OWNER_TAG>
  <CUSTOMIZATION_LEVEL>L</CUSTOMIZATION_LEVEL>
  <LICENSED_FLAG>Y</LICENSED_FLAG>
  <DISPLAY_NAME>OM Generic Inbound Event</DISPLAY_NAME>
  <DESCRIPTION>OM Generic Inbound Event</DESCRIPTION>
  <IREP_ANNOTATION>/*# * This event is raised after the Purchase Order
has been pushed to Oracle Order management open interface tables. This
event will start the workflow OEOI/R_OEOI_ORDER_IMPORT to import the
order. * * @rep:direction OUT * @rep:scope public * @rep:displayname OM
```

```
Generic Inbound Event * @rep:lifecycle active * @rep:product ONT * @rep:
compatibility S * @rep:category BUSINESS_ENTITY ONT_SALES_ORDER
*/</IREP_ANNOTATION>
  </WF_EVENTS>
  </WF_TABLE_DATA>
  </oracle.apps.wf.event.all.sync>
```

Business Entity Annotation Guidelines

Business entities are things that either perform business activities or have business activities performed on them. Account numbers, employees, purchase orders, customers, and receipts are all examples of business entities.

What Is the Importance of Business Entities?

Business entities are highly desired search criteria in the context of the Integration Repository. The design of the Integration Repository UI includes "browse by business entity" functionality.

Where Do Business Entities Appear in Repository Annotations?

The `rep:category BUSINESS_ENTITY` annotation is where you associate a given interface with a business entity. For a general description of the `rep:category` annotation, see rep:category, page A-128.

> **Note:** In certain cases where the entity's display name itself is sufficiently self-descriptive, it can serve as the description as well.

Existing Business Entities

Custom integration interfaces can use only seeded or existing business entities.

> **Note:** Integration Repository currently does not support the creation of custom Product Family and custom Business Entity.

The following table lists the existing business entities:

List of Business Entities

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
AHL_DOCUMENT	Document	Electronic Document or Document Reference
AHL_ITEM_COMPOSITION	Tracked Item Composition	It is the list of item groups or non-tracked items that a tracked item is composed of.

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
AHL_ITEM_GROUP	Alternate Item Group	A group of similar items where one can be interchanged for another while performing maintenance.
AHL_MAINT_OPERATION	Maintenance Operation	It defines resource and material requirements. It is basic definition of work.
AHL_MAINT_REQUIREMENT	Maintenance Requirement	It is maintenance requirement definition. It defines routes, applicability on item or unit instances. It also defines frequency based on time and counters.
AHL_MAINT_ROUTE	Maintenance Route	It contains set of operations, and defines dispositions, resource and material requirements.
AHL_MAINT_VISIT	Maintenance Visit	It connects an unit or item instance with a block of tasks. It is an organization and department where the maintenance work takes place, and when the work is to be accomplished.
AHL_MAINT_WORKORDER	Maintenance Workorder	Maintenance Workorder with a schedule
AHL_MASTER_CONFIG	Master Configuration	A Master Configuration models the structure of an electromechanical system assembly.
AHL_OSP_ORDER	Outside Service Order	An order that contains the information required to service parts by a third party organization.

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
AHL_PROD_CLASS	Product Classification	It is the categorization of units or items pertaining to maintenance and usage.
AHL_UNIT_CONFIG	Unit Configuration	An Unit Configuration describes the structure of an assembled electromechanical system.
AHL_UNIT_EFFECTIVITY	Unit Maintenance Plan Schedule	Unit Maintenance Plan with a due date
AHL_UNIT_SCHEDULES	Unit Usage Event	Event describes usage of a configured unit for a specific time period, such as an airplane flight.
AME_ACTION	Approval Action	Approval Action specifies an action to be performed, if the conditions of an approval rule is satisfied. For example, 'Require approvals up to the first three superiors'.
AME_APPROVAL	Approval	Approval
AME_APPROVER_GROUP	Approvals Management Approver Group	A predefined group of approvers who will be assigned to approve actions of specific business processes/transactions.
AME_APPROVER_TYPE	Approver Type	Classification of approvers who can be used in Approvals Management. For example, all HR employees are classified as the approver type as PER in Approvals Management.

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
AME_ATTRIBUTE	Approvals Management Attribute	Object to capture business attributes for a transaction which requires approval. For example, INVOICE_AMOUNT can be an attribute which captures the total amount of an invoice.
AME_CONDITION	Approval Rule Condition	Condition based on the Approvals Management attribute that evaluates the approval rules. An example of condition on the attribute INVOICE_AMOUNT can be "INVOICE_AMOUNT > 10,000 USD".
AME_CONFIG_VAR	Approval Configuration Variable	A set of approval configurations which controls certain behavior within Approvals Management.
AME_ITEM_CLASS	Approvals Management Item Class	It is the classification of certain Approval Management objects into different classes like Header, Line Item, Cost Center.
AME_RULE	Approvals Business Rule	Approval Business rule consisting of a set of conditions, when satisfied, will dictate some actions to happen (which will result in a list of approvers).
AME_TRANSACTION_TYPE	Approval Transaction Type	A set of approval attributes, conditions, and rules making up a approval policy.
AMS_BUDGETS	Marketing Budget	It is the budget for Marketing Campaigns, Events, and other marketing activities.

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
AMS_CAMPAIGN	Marketing Campaign	Marketing Campaign
AMS_EVENT	Marketing Event	Marketing Event
AMS_LEAD	Sales Lead	Sales Lead
AMS_LIST	Marketing List	Marketing List
AMS_METRIC	Marketing Metric	It is a measurement of marketing operations, such as, number of responses generated by a campaign.
AP_INVOICE	Payables Invoice	Payables Invoice
AP_PAYMENT	Supplier Payment	Supplier Payment
AP_PAYMENT_ADVICE	Payment Advice	Payment Advice
AP_SUPPLIER	Supplier	Supplier
AP_SUPPLIER_CONTACT	Supplier Contact	Supplier Contact
AP_SUPPLIER_SITE	Supplier Site	Supplier Site
AR_ADJUSTMENT	Receivables Invoice Adjustment	Receivables Invoice Adjustment
AR_BILLS_RECEIVABLE	Bills Receivable	Bills Receivable
AR_CHARGEBACK	Chargeback	Chargeback
AR_CREDIT_MEMO	Credit Memo	Credit Memo
AR_CREDIT_REQUEST	Credit Request	Credit Request
AR_DEBIT_MEMO	Debit Memo	Debit Memo
AR_DEPOSIT	Deposit	Deposit

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
AR_INVOICE	Receivables Invoice	Receivables Invoice
AR_PREPAYMENT	Prepayment	Prepayment
AR_RECEIPT	Receivables Receipt	Receivables Receipt
AR_REMITTANCE	Remittance	Remittance
AR_REVENUE	Revenue	Revenue
AR_SALES_CREDIT	Sales Credit	Sales Credit
AR_SALES_TAX_RATE	Sales Tax Rate	Sales Tax Rate
ASN_OPPORTUNITY	Sales Opportunity	Sales Opportunity
ASN_SALES_TEAM	Sales Team	Sales Team on an Opportunity or an Account, or a Lead
ASO_QUOTE	Sales Quote(1)	A sales quote is a business object that contains detailed information on the products, prices, terms, etc. in the solution proposed to potential customers(1).
AS_OPPORTUNITY	Sales Opportunity(1)	Sales Opportunity(1)
BEN_CWB_3RD_PARTY_STOCK_OPTS	Third Party Stock Option	Third Party Stock Options
BEN_CWB_AUDIT	Compensation Workbench Audit	It records every change event within a Compensation Workbench user session. This covers all compensation elements.
BEN_CWB_AWARD	Compensation Workbench Award	It is an employee monetary award. For example, salary raise, salary bonus, or shares.

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
BEN_CWB_BUDGET	Compensation Workbench Budget	it is the budget of money or shares available for a manager to distribute including base salaries and bonuses.
BEN_CWB_PERSON	Compensation Workbench Person	Snapshot of a HR person on a specific date, for Compensation Workbench processing.
BEN_CWB_PLAN	Compensation Workbench Plan	It is a Compensation Plan, such as Salary Raise Plan, Bonus Plan or Stock Option Plan.
BEN_CWB_TASK	Compensation Workbench Task	It is the task performed in managing a Compensation Workbench Plan. For example, budgeting, allocation of amounts, submitting work and approval.
BIS_REPORT	BIS Report	BIS Report
BOM_BILL_OF_MATERIAL	Bill of Material	This interface adds, changes, and deletes Bill of Material of any type.
BOM_MFG_ROUTING	Product Manufacturing Routing	A routing defines the step-by-step operations required to produce an assembly in accordance with its Bill of Material.
BOM_PRODUCT_FAMILY	Product Family	Product Family for Planning Purposes
CAC_APPOINTMENT	Appointment	Appointment or Meeting for a given date and time period

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
CAC_BUSINESS_OBJECT_META_DATA	Business Object Meta Data Definition	Metadata definition for a Business Entity. It is used to dynamically link to external business entities. Also it is used for querying entity details, building dynamic LOVs and search pages.
CAC_CAL_TASK	Calendar Task	Task that will appear on User's Calendar as a time Blocking Task or a Todo.
CAC_NOTE	Note	Notes or Comments associated to different Business Objects
CAC_RS_TIME_BOOKING	Resource Time Booking	Time Booking for Person and non Person (e.g. Conference room) Resources
CAC_SCHEDULE	Schedule	Schedule
CAC_SCHEDULE_TEMPLATE	Schedule Template	Schedule Template
CAC_SYNC_SERVER	Calendar Synchronization Server	Calendar server to synchronize calendar entities like Task, Appointments, Contacts etc. to external calendars.
CAC_TASK_TEMPLATE	Calendar Task Template	Calendar Task Template
CCT_ADVANCED_TELEPHONY_SDK	Advanced Telephony SDK	This SDK allows telephony integration with Oracle E-Business Suite using server side integration.
CCT_BASIC_TELEPHONY_SDK	Basic Telephony SDK	This SDK allows telephony integration with Oracle E-Business Suite using client side integration.

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
CE_BANK_STATEMENT	Bank Statement	Bank Statement
CE_RECONCILIATION_ITEM	Reconciliation Item	Reconciliation Item
CHV_PLANNING_SCHEDULE	Buyer Forecast	Buyer Forecast
CHV_SHIPPING_SCHEDULE	Buyer Shipment Request	Buyer Shipment Request
CLN_TRADING_PARTNER_COLL	Collaboration Trading Partner	Trading Partner
CLN_TRADING_PARTNER_COLL_EVENT	Trading Partner Collaboration Event	Trading Partner Collaboration Event
CN_COMP_PLANS	Incentive Compensation Plan	Incentive Compensation Plan
CN_INCENTIVES	Incentive Compensation	Variable compensation or rebates that can be monetary or non-monetary rewards for sales people, partners or customers.
CSD_REPAIR_ESTIMATE	Repair Estimate	Repair Estimate shows the total cost for the repair execution, which can include material, labor and expense charge lines.
CSD_REPAIR_LOGISTICS	Repair Logistics	Repair Logistics track the receiving and shipping of the customer item being repaired and also the items being loaned.
CSD_REPAIR_ORDER	Repair Order(1)	Repair Order(1)
CSF_TASK_DEBRIEF	Service Task Debrief	Service task debrief of material, labor and expense

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
CSI_COUNTER	Counters	It provides a mechanism to define and maintain different types of Matrixes. These can be attached to objects in the Oracle E-Business Suite like Installed Base Instances, or Service Contract Lines.
CSI_ITEM_INSTANCE	Item Instance	Install Base Item Instance
CST_DEPARTMENT_OVERHEAD	Manufacturing Department Overhead Rate	Manufacturing Department Overhead Rate
CST_ITEM_COST	Inventory Item Cost	Inventory Item Cost
CST_RESOURCE_COST	Manufacturing Resource Unit Cost	Manufacturing Resource Unit Cost
CS_SERVICE_CHARGE	Service Charge	Service Charge
CS_SERVICE_REQUEST	Service Request	Service Request
CZ_CONFIG	Configuration	Configuration
CZ_CONFIG_MODEL	Configuration Model	Configuration Model
CZ_MODEL_PUB	Configuration Model Publication	Configuration Model Publication
CZ_RP_FOLDER	Configurator Repository Folder	Configurator Repository Folder
CZ_USER_INTERFACE	Configuration Model User Interface	Configuration Model User Interface
DPP_EXECUTION_REQUEST	Execution Integration Request	It is an entity for integration of DPP with other Applications. It is used by event invoked from DPP UI and concurrent programs for integration with external applications like AR, AP, etc.

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
DPP_TRANSACTION_APPROVAL	Transaction Approval Notification	This entity is defined for the AME Approval for DPP transaction. It is referenced in UI on clicking of the **Request Approval** button in a New DPP transaction.
DPP_XMLG_OUTBOUND	Outbound pre-approval process	It is an entity used by events to trigger preapproval process through Oracle XML Gateway for Price Protection.
EAM_ASSET_ACTIVITY_ASSOCIATION	Maintenance Asset Activity Association	Maintenance Asset Activity Association
EAM_ASSET_ACTIVITY_SUPPRESSION	Asset activity suppression relations	It indicates that an asset preventive maintenance activity is suppressed due to the performance of another activity.
EAM_ASSET_AREA	Maintenance Asset Area	Maintenance Asset Area
EAM_ASSET_ATTRIBUTE_GROUPS	Maintenance Asset Attribute Group	Maintenance Asset Attribute Group
EAM_ASSET_ATTRIBUTE_VALUE	Maintenance Asset Attribute Value	Maintenance Asset Attribute Value
EAM_ASSET_METER	Maintenance Asset Meter Association	Maintenance Asset Meter Association
EAM_ASSET_NUMBER	Maintenance Asset Number	Maintenance Asset Number
EAM_ASSET_ROUTE	Maintenance Asset Route	Maintenance Asset Route
EAM_COMPLETE_WO_OPERATION	Maintenance Work Completion	Maintenance Work Completion
EAM_DEPARTMENT_APPROVER	Maintenance Department Approver	Maintenance Department Approver - User or responsibility

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
EAM_METER	Meter	Meter
EAM_METER_READING	Meter Reading	Meter Reading
EAM_PARAMETER	Maintenance Setup	Maintenance Setup
EAM_PM_SCHEDULE	Preventive Maintenance Schedule	Preventive Maintenance Schedule
EAM_SET_NAME	Maintenance Set	Maintenance Set
EAM_WORK_ORDER	Asset Maintenance Work Order	Asset Maintenance Work Order
EAM_WORK_REQUEST	Maintenance Work Request	Maintenance Work Request
ECX_CONFIRM_BOD	XML Gateway Confirmation Message	XML Gateway Confirmation Message
ECX_MESSAGE_DELIVERY	XML Gateway Message Delivery	It is used by both Oracle and non Oracle messaging systems to report delivery status. Status information is written to XML Gateway log tables to track and report transaction delivery data.
ECX_TRADING_PARTNER	XML Gateway Trading Partner	It represents a business partner at a particular address with whom you exchange business messages. It could be a customer, supplier, bank branch, or an internal location.
ECX_TRANSFORMATION	XML Gateway Transformation	This interface is used to apply a style sheet to an XML message and return the transformed XML message for further processing by the calling environment.

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
EC_CODE_CONVERSION	Code Conversion	It converts Oracle's Internal Codes to External System Codes and vice-versa, such as Currency Code, Unit Of Measure.
EC_EDI_TRANSACTION_LAYOUT	EDI Transaction Layout Definition Report	EDI Transaction Layout Definition Report
EC_INBOUND	Inbound EDI Message	It is an EDI message sent to the system from a trading partner.
EC_OUTBOUND	Outbound EDI Message	It is an EDI message sent from the system to a trading partner.
EC_TP_MERGE	Trading Partner Merge	It indicates a merge of Trading Partners as a result of an account merge in the Trading Community Architecture (TCA).
EDR_EVIDENCE_STORE	E-Records Evidence Store	E-Records Evidence Store
EDR_ISIGN_FILE_UPLOAD	File Upload Approval Request	File Upload Approval Request
EGO_ITEM	Catalog Item	An item that is listed in the Item Catalog.
EGO_USER_DEFINED_ATTR_GROUP	PLM User Defined Attributes	This interface adds, changes, deletes, and queries User-defined attributes for any entity.
ENG_CHANGE_ORDER	Product Change Order	Product or Engineering Change
FA_ASSET	Asset	The interface for adding assets to Oracle Assets.

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
FA_CAPITAL_BUDGET	Capital Budget	The interface for uploading capital budgets to Oracle Assets.
FA_LEASE_PAYMENT	Lease Payment	The interface for sending lease payment lines to Oracle Payables.
FEM_ACCOUNT_FACT	Analytic Account Information	Detail level financial account data
FEM_BALANCES_FACT	Analytic Balances	It includes Ledger input and Ledger Profitability processing results.
FEM_FACT_REPOSITORY	Enterprise Analytical Fact Repository	It contains numeric facts (often called measurements) that can be categorized by multiple dimensions. It contains either detail-level facts or facts that have been aggregated.
FEM_STATISTICAL_FACT	Analytic Statistical Information	It contains dimensional numerical measures. These measures are actual statistical values, both derived and empirically obtained.
FEM_TRANSACTION_FACT	Analytic Transaction Information	The information represents counts of events and interactions for financial accounts.
FEM_XDIM_ACTIVITY	Analytic Activity	It describes repeatable tasks in relation to other dimensions. It is defined by an action and acted upon item. Business processes and actions of individuals can be categorized as activities.

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
FEM_XDIM_AUXILIARY	Auxiliary Analytic Dimensions	It indicates the "non-foundation" dimensions for the Enterprise Performance Foundation. Unlike Foundation dimensions, they are not employed by calculation engines for value-added processing.
FEM_XDIM_BUDGET	Analytic Budget	It identifies budgets and forecasts.
FEM_XDIM_CAL_PERIOD	Analytic Calendar Period	Analytic Calendar Period
FEM_XDIM_CCTR_ORG	Analytic Organization	It indicates Standard Analytic Organization dimension made up of Company and Cost Center.
FEM_XDIM_CHANNEL	Analytic Channel	It identifies distribution and sales channels.
FEM_XDIM_COMPANY	Company Dimension	Standard Analytic Company dimension
FEM_XDIM_COST_CENTER	Cost Center Dimension	Standard Analytic Cost Center dimension
FEM_XDIM_COST_OBJECT	Analytic Cost Object	A Cost Object is a multidimensional entity that describes a cost.
FEM_XDIM_CUSTOMER	Analytic Customer	It identifies groups or individuals with a business relationship to analytic data.
FEM_XDIM_DATASET	Analytic Dataset	It identifies generic containers for analytic data.
FEM_XDIM_ENTITY	Analytic Consolidation Entity	It identifies Consolidation, Elimination and Operating Entities for Global Consolidation System Users.

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
FEM_XDIM_FINANCIAL_ELEM	Analytic Financial Element	It identifies categories of amount types for balances, statistics and rates.
FEM_XDIM_GENERIC_FACT_DATA	Analytic User Defined Fact Data	Tables available for storing fact data of user defined dimensionality
FEM_XDIM_GEOGRAPHY	Analytic Geography	It identifies geographic locations.
FEM_XDIM_HIERARCHY	Analytic Dimension Hierarchy	It is organized parent-child relationships of dimension members.
FEM_XDIM_LEDGER	Analytic Ledger	It identifies books of account. It is analogous to a Set of Books.
FEM_XDIM_LEVEL	Analytic Dimension Level	It identifies categories for dimension members.
FEM_XDIM_LINE_ITEM	Analytic Line Item	It identifies general ledger accounts, typically as an extension to Natural Accounts.
FEM_XDIM_NATURAL_ACCOUNT	Analytic Natural Account	It identifies an account within an organization where balances are posted for the five different balance types of revenue, expense, owners equity, asset and liability.
FEM_XDIM_PRODUCT	Analytic Product	It identifies commodities or services offered for sale.
FEM_XDIM_PROJECT	Analytic Project	It identifies plans and endeavors.
FEM_XDIM_SIC	Analytic Standard Industrial Classification	It identifies official codes of the Standard Industrial Classification system.

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
FEM_XDIM_SIMPLE	Analytic List of Values only Dimension	Grouping of all Analytic dimensions that have no attributes and serve only as lists of values.
FEM_XDIM_SOURCE_SYSTEM	Analytic Source System	It identifies the point of origin for fact and dimension data.
FEM_XDIM_TASK	Analytic Task	It identifies individual operations and pieces of work.
FEM_XDIM_USER_DIMENSION	Analytic User Defined Dimension	It is the grouping of all customizable analytic attributed dimensions.
FF_FORMULA_FUNCTION	Fast Formula Function	It represents an external procedural call providing arbitrary extensions to core Fast Formula functionality.
FLM_FLOW_SCHEDULE	Flow Schedule	Flow Schedule
FND_APPS_CTX	Oracle E-Business Suite Applications Security Context	Applications context representing current user session
FND_CP_PROGRAM	Concurrent Program	Discrete unit of work that can be run in the concurrent processing system. Typically, a concurrent program is a long-running, data-intensive task, such as generating a report.
FND_CP_REQUEST	Concurrent Request	It is the request to the concurrent processing system to run a program with a given set of parameter values, an optional schedule to repeat, and optional postprocessing actions.

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
FND_CP_REQUEST_SET	Concurrent Request Set	A convenient way to run several concurrent programs with predefined print options and parameter values. Request sets group requests into stages that are submitted by the set.
FND_EBS_MOBILE	Mobile Optimized API	Mobile optimized APIs are light-weight APIs designed and built for Oracle E-Business Suite mobile app development.
FND_FLEX_KFF	Key Flexfield	Customizable multi-segment fields
FND_FORM	Oracle E-Business Suite Applications Form	A form is a special class of function that you may navigate to them using the Navigator window.
FND_FUNCTION	Oracle E-Business Suite Applications Function	A function is a part of an application functionality that is registered under an unique name for the purpose of providing function security.
FND_FUNC_SECURITY	Function Security	Function security restricts application functionality to authorized users.
FND_GFM	Oracle E-Business Suite Applications File	Generic file manager provides ways to upload/download files and manipulate the file attributes.
FND_LDAP_OPERATIONS	LDAP Directory	Enable Oracle E-Business Suite to performs operations against the integrated OID.

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
FND_MBL_SAMPLE	Sample Mobile Interfaces	Sample interfaces used by Oracle E-Business Suite Mobile Foundation's sample mobile app. These interfaces are not designed for production use, but used only for demonstration purposes.
FND_MENU	Oracle E-Business Suite Applications Menu	A hierarchical arrangement of functions and menus of functions that appears in the Navigator.
FND_MESSAGE	Oracle E-Business Suite Applications Message Dictionary	It contains catalog / repository of messages for the entire Oracle E-Business Suite. Message Dictionary facility is used to display and logging from application.
FND_NAVIGATION	Oracle E-Business Suite Applications Navigation	Standard ways of navigating from one page to another within applications
FND_OBJECT_CLASSIFICATION	OATM Object-Tablespace Classification	This entity stores seeded, explicit OATM object-tablespace classifications, which can be further customized.
FND_PROFILE	User Profile	It is a set of changeable options that affects the way the application behaves run time.
FND_RESPONSIBILITY	Responsibility	A responsibility defines the menu structure for a product in Oracle E-Business Suite.
FND_SSO_MANAGER	Single Sign On Manager	Single Sign On and Central Login related APIs

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
FND_TABLESPACE	Tablespace Model Tablespace	It classifies all storage-related objects. Logical Tablespaces have a 1:1 relation with physical tablespaces.
FND_USER	User	It represents a user of Oracle E-Business Suite.
FUN_ARAP_NETTING	Payables and Receivables Netting	Payables and Receivables for Netting
FUN_IC_TRANSACTION	IC Manual Transaction	Intercompany transaction will be between one initiator and single/multiple recipients.
FUN_INTERCOMPANY_BATCH	Intercompany Transaction Set	It is intercompany batch containing transactions between legal entities.
FV_BUDGETARY_DISCOUNT	Federal Budgetary Discount	It creates Budgetary Discount Transactions.
FV_BUDGET_JOURNAL	Federal Budget Execution Document	It contains federal budget records imported into federal budgetary tables.
FV_FINANCE_CHARGE	Federal Finance Charge	Federal Finance Charge
FV_IPAC_DISBURSEMENT	IPAC Disbursement	IPAC Disbursement
FV_PRIOR_YEAR_ADJUSTMENT	Prior Year Adjustment	Prior Year Adjustment
FV_TREASURY_DISBURSEMENT	Treasury Disbursement	Treasury Confirmation, Backout and Void Disbursement Transactions
FV_YEAR_END_CLOSE	Federal Year End Closing Information	Federal Year End Closing
GHR_DUTY_STATION	US Federal Workplace Duty Station	US Federal Workplace Duty Station

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
GHR_EEO_COMPLAINT	US Federal EEO Complaint	US Federal EEO Complaint
GHR_POSITION_DESCRIPTION	Position Description	Position Description
GHR_REQ_FOR_PERSONNEL_ACTION	Request for Personnel Action	Request for Personnel Action
GL_ACCOUNTING_SETUP_MANAGER	Accounting Setup Manager	This represents the Accounting Setup of Ledgers and Legal Entities in General Ledger.
GL_ACCOUNT_COMBINATION	General Ledger Code Combination	This represents General Ledger Account Combinations Defined Under Chart of Accounts.
GL_BC_PACKETS	Budgetary Fund Control Transaction Packet	Budgetary Fund Control Transaction Packet
GL_BUDGET_DATA	General Ledger Budget Data	General Ledger Budget Data
GL_CHART_OF_ACCOUNTS	Chart of Accounts	Chart of Accounts (COA)
GL_DAILY_RATE	Daily Currency Conversion Rate	Daily Currency Conversion Rate
GL_INTERCOMPANY_TRANSACTION	Intercompany Transaction	Intercompany Transaction
GL_JOURNAL	Journal Entry	Journal Entry
GL_PERIOD	General Ledger Accounting Period	This represents the Accounting Period defined in Accounting Calendar.
GMD_ACTIVITIES_PUB	Product Development Activity	It creates, modifies, or deletes activity information.

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
GMD_FORMULA	Process Manufacturing Formula	Process Manufacturing Formula
GMD_OPERATION	Process Manufacturing Operation	Process Manufacturing Operation
GMD_OUTBOUND_APIS_PUB	Process Manufacturing Quality Outbound Transaction	It is public level Process Manufacturing Quality package containing APIs to export information to third party products.
GMD_QC_SAMPLES	Process Manufacturing Quality Sample	Process Manufacturing Quality Sample
GMD_QC_SPEC	Process Manufacturing Quality Specification	Process Manufacturing Quality Specification
GMD_QC_SPEC_VR	Process Manufacturing Specification Usage Rule	Process Manufacturing Specification Usage Rule
GMD_QC_TESTS_PUB	Process Manufacturing Quality Test	Process Manufacturing Quality Test
GMD_RECIPE	Process Manufacturing Recipe	Process Manufacturing Recipe
GMD_RECIPE_VALIDITY_RULE	Process Manufacturing Recipe Usage Rule	Process Manufacturing Recipe Usage Rule
GMD_RESULTS_PUB	Process Manufacturing Quality Test Result	Process Manufacturing Quality Test Result
GMD_ROUTING	Process Manufacturing Routing	Process Manufacturing Routing
GMD_STATUS_PUB	Process Manufacturing Product Development Status	It modifies the status for routings, operations, receipts, and validity rules.
GME_BATCH	Process Manufacturing Batch	Process Manufacturing Batch

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
GME_BATCH_STEP	Process Manufacturing Batch Step	Process Manufacturing Batch Step
GMF_ALLOCATION_DEFINITION	Process Manufacturing Expense Allocation Definition	It is the setup data for allocating indirect expenses (indirect overheads) to items.
GMF_BURDEN_DETAIL	Process Manufacturing Financials Overhead Detail	It indicates overhead costs assigned to items that have been manufactured or purchased.
GMF_ITEM_COST	Process Manufacturing Financials Item Cost	Process Manufacturing Financials Item Cost
GMF_RESOURCE_COST	Process Manufacturing Financials Resource Cost	Process Manufacturing Financials Resource Cost
GMI_ADJUSTMENTS	Process Manufacturing Inventory Adjustment	Process Manufacturing Inventory Adjustment
GMI_API	Process Manufacturing Inventory Setup	It is the Process Manufacturing Inventory transaction to create, modify, delete items, lots, lot conversions.
GMI_ITEM	Process Manufacturing Item	Process Manufacturing Item
GMI_ITEM_LOT_UOM_CONV	Process Manufacturing Item Lot UOM Conversion	Process Manufacturing Item Lot UOM Conversion
GMI_LOT	Process Manufacturing Lot	Process Manufacturing Lot
GMI_OM_ALLOC_API_PUB	Process Manufacturing Sales Order Inventory Allocation	The Allocate OPM Orders API is a business object that can create, modify, or delete OPM reservation (allocation) information for Order Management.

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
GMI_PICK_CONFIRM_PUB	Process Manufacturing Sales Order Inventory Pick Confirmation	The Pick Confirm API is a business object that pick confirms, or stages the inventory for a Process Move Order Line or a Delivery Detail line.
GMP_CALENDAR_API	Process Planning Shop Calendar	It modifies the Shop Calendar.
GMP_GENERIC_RESOURCE	Generic Process Manufacturing Resource	Manufacturing resource in Process Manufacturing
GMP_PLANT_RESOURCE	Process Manufacturing Plant Resource	Plant specific manufacturing resource in Process Manufacturing
GMP_RSRC_AVL_PKG	Process Planning Resource Availability	It modifies resource availability.
GMS_AWARD	Project Award Budget	Project Award Budget
HR_AUTHORIA_INTEGRATION_MAP	Authoria Integration Map	Authoria Integration Map
HR_BUDGET	HR Budget	HR Budget
HR_BUSINESS_GROUP	Business Group	Business Group
HR_CALENDAR_EVENT	HR Calendar Event	HR Calendar Event
HR_COST_CENTER	Cost Center	Cost Center
HR_EVENT	HR Bookable Event	HR Bookable Event
HR_HELP_DESK	HR Help Desk Integration	Peoplesoft Help Desk Integration points with the Oracle E-Business Suite HRMS
HR_KI_MAP	Knowledge Integration Map	Knowledge Integration Map

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
HR_KI_SYSTEM	Knowledge Integration System	Knowledge Integration System
HR_LEGAL_ENTITY	Legal Entity	Legal Entity
HR_LIABILITY_PREMIUM	Liability Premium	Liability Premium
HR_LOCATION	Location	Location
HR_MESSAGE_LINE	HRMS Message Line	HRMS Message Line
HR_OPERATING_UNIT	Operating Unit	Operating Unit
HR_ORGANIZATION	HRMS Organization	HRMS Organization
HR_ORGANIZATION_LINK	Organization Link	Organization Link
HR_PAY_SCALE	Pay Scale	Pay Scale
HR_PERSON	HR Person(1)	HR Person(1)
HR_PERSONAL_DELIVERY_METHOD	Personal Delivery Method	Personal Delivery Method
HR_ROLE	HRMS Role	HRMS Role
HR_SALARY_BASIS	Salary Basis	Salary Basis
HR_SELF_SERVICE_TRANSACTION	HR Self Service Transaction	Self Service Transaction
HR_SOC_INS_CONTRIBUTIONS	Social Insurance Contribution	Social Insurance Contribution
HR_SUPER_CONTRIBUTION	Superannuation Contribution	It indicates payment to a fund providing for a person's retirement.
HR_USER_HOOK	HRMS User Hook	HRMS User Hook
HXC_TIMECARD	Timecard	Timecard

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
HXC_TIMECARD_RECURRING_PERIOD	Timecard Recurring Period	Timecard Recurring Period
HXC_TIME_INPUT_SOURCE	Time Input Source	It indicates how Timecard data was input.
HXC_TIME_RECIPIENT	Time Recipient Application	An application that receives and processes Time and Labor Data.
HZ_ACCOUNT_CONTACT	Customer Account Contact	A person who is the contact for a customer account.
HZ_ADDRESS	Trading Community Address	It is an address of a trading community member, for example, a customer's or partner's address.
HZ_CLASSIFICATION	Trading Community Classification	It is a categorization of parties, using user-defined or external standards such as the NAICS, NACE, or SIC.
HZ_CONTACT	Trading Community Contact	A person who is a contact for an organization or another person.
HZ_CONTACT_POINT	Contact Point	It is a means of contact, for example, phone or e-mail.
HZ_CONTACT_PREFERENCE	Contact Preference	It is the information about when and how parties prefer to be contacted.
HZ_CUSTOMER_ACCOUNT	Customer Account	A person or organization that the deploying company has a selling relationship with.
HZ_EXTERNAL_REFERENCE	Trading Community External Reference	Management of operational mappings between the trading community database and external source systems.

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
HZ_GROUP	Trading Community Group	Trading Community Group
HZ_ORGANIZATION	Trading Community Organization	It is a party of type Organization and related information, including financial and credit reports.
HZ_PARTY	Party	A trading community entity, either person or organization, that can enter into business relationships.
HZ_PERSON	Trading Community Person	It is a party of type Person and related information, such as employment and education.
HZ_RELATIONSHIP	Trading Community Relationship	A representation of how two parties are related, based on the role that each party plays with respect to the other.
HZ_RELATIONSHIP_TYPE	Trading Community Relationship Type	A categorization of roles that parties can play in relationships.
IBC_CONTENT_DELIVERY_MANAGER	Content Delivery Manager	Content Delivery Manager class provides APIs for applications to retrieve content items stored in the OCM Content Repository.
IBE_CATALOG_PUNCHOUT	Web Store Catalog Punchout	It is a process of enabling procurement users to choose items available in iStore catalog. The login/logout of procurement users in iStore is transparent to them.
IBE_CONTENT	Web Store Content	Web Store Page Content
IBE_ITEM	Web Store Item	Web Store Product Item

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
IBE_SALES_ORDER	Web Store Sales Order	Web Store Sales Order
IBE_SECTION	Web Store Section	Navigational Hierarchy for Web content and product
IBE_SESSION_ATTRIBUTES	Web Store Session Attributes	Session Attributes of Users visiting the Web Store
IBE_SHOPPING_CART	Web Store Shopping Cart	Web Store Shopping Cart
IBE_SHOPPING_LIST	Web Store Shopping List	Web Store Shopping List
IBE_SITE	Web Store Site	Web Store Site
IBE_TEMPLATE	Web Store Template	Web Store Page Template
IBE_USER	Web Store User	Users, Contacts, Customers
IBW_PAGE_ACCESS_TRACKING	Web Analytics Page Access Tracking	It captures visit and page access data required for Web analytics reporting.
IBY_BANKACCOUNT	External Bank Account	Supplier or Customer Bank Account
IBY_CREDITCARD	Credit Card	Credit Card Payment Instrument
IBY_EXCEPTION	IBY Exception	It is an exception generated by IBY code when an error is encountered.
IBY_FUNDSCAPTURE_ORDER	Funds Capture Order	It is a single funds capture request delivered to a payment system by the request payee.
IBY_PAYMENT	IBY Payment	It indicates payment made through IBY to the supplier.

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
IEO_AGENT	Interaction Center Agent	A person that interacts with a customer during an interaction event.
IEX_COLLECTION_CASE	Collection Case	Collection Case
IEX_COLLECTION_DISPUTE	Collection Dispute	A dispute creates a credit memo request in Oracle Receivables to resolve all or part of an invoice that a customer contends is not owed.
IEX_COLLECTION_PROMISE	Collection Promise	Collection Promise
IEX_COLLECTION_SCORE	Collection Score	Collection Score
IEX_COLLECTION_STRATEGY	Collection Strategy	Collection Strategy
IEX_PROMISES	Collection Payment Promise	A promise to pay is a non-binding agreement from the customer to make a payment at a certain date.
IEX_STRATEGY	Receivables Collection Strategy	Strategies are a pre-configured sequence of work items that automate the process of collecting open receivables and support complex collections management activities.
IGC_CONTRACT_COMMITMENT	Contract Commitment	Contract Commitment
IGC_ENCUMBRANCE_JOURNAL	Encumbrance Journal	Encumbrance Journal
IGF_AWARD	Financial Aid Student Award	Financial Aid Student Award

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
IGF_BASE_RECORD	Financial Aid Student Base Record	Financial Aid Student Base Record
IGF_COA	Student Attendance Cost	Student Attendance Cost
IGF_DL	Financial Aid Direct Loan	Financial Aid Direct Loan
IGF_FFELP	Financial Aid FFELP Loan	Financial Aid FFELP Loan
IGF_FWS	Financial Aid Work Study	Financial Aid Work Study
IGF_ISIR	Institutional Student Information Record	Institutional Student Information Record
IGF_PELL	Financial Aid Pell Grant	Financial Aid Pell Grant
IGF_PROFILE	Student Profile Application	Student Profile Application
IGF_TODO	Financial Aid Student Todo Item(1)	Financial Aid Student Todo Item(1)
IGF_VERFN	Financial Aid Verification Item	Financial Aid Verification Item
IGS_ADM_APPLICATION	Admission Application	Admission Application
IGS_ADM_FEE	Admission Fee	Admission Application Fee
IGS_ADV_STAND	Advanced Standing	Advanced Standing
IGS_DA_REQUEST	Degree Audit Request	Degree Audit Request
IGS_INQ_APPLICATION	Prospective Applicant Inquiry	Prospective Applicant Inquiry
IGS_INSTITUTION	Institution	Institution Party
IGS_PARTY_CHARGE	Higher Education Party Charge	Higher Education Party Account Charge Transactions
IGS_PARTY_CREDIT	Higher Education Party Credit	Higher Education Party Account Credit Transactions

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
IGS_PARTY_REFUND	Higher Education Party Refund	Higher Education Party Account Refund Transactions
IGS_PERSON_ALTERNATE_ID	Alternate Person Identifier	Person Alternate Identifier e. g. SSN, Driver Licence etc
IGS_PERSON_CONTACT	Person Contact Information	Person Contact Information
IGS_PREV_EDUCATION	Previous Education	Previous Education
IGS_PROGRAM	Higher Education Program	Higher Education Program
IGS_SPONSORSHIP	Student Sponsor Relationship	Student Sponsor Relationship
IGS_STUDENT_CONCENTRATION	Student Concentration	Student Concentration
IGS_STUDENT_PROGRAM	Student Program Attempt	Student Program Attempt
IGS_STUDENT_UNIT	Student Unit Attempt	Student Unit Attempt
IGS_TODO	Financial Aid Student Todo Item	Financial Aid Student Todo Item
IGS_UNIT	Higher Education Unit	Higher Education Unit
IGW_PROPOSAL	Grants Proposal	Grants Proposal
IGW_PROPOSAL_BUDGET	Grants Proposal Budget	Grants Proposal Budget
INV_ACCOUNTING_PERIOD	Inventory Accounting Period	Status of an inventory accounting period
INV_ALLOCATION	Material Allocation	Inventory Material Allocation
INV_CONSIGNED_DIAGNOSTICS	Consigned Inventory Diagnostics	Set of utilities that identify and communicate inaccuracies in setup data of Consigned Inventory from Supplier feature.

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
INV_COUNT	Material Count	Material Count
INV_IC_TRANSACTION_FLOW	Inventory Intercompany Invoicing Transaction Flow	It is an execution of the transactions that generate intercompany invoices in Inventory.
INV_IC_TRANSACTION_FLOW_SETUP	Intercompany Inventory Transaction Flow Setup	Intercompany Inventory Transaction Flow Setup
INV_LOT	Inventory Lot	Inventory Lot
INV_MATERIAL_TRANSACTION	Material Transaction	Inventory Material Transaction
INV_MOVEMENT_STATISTICS	Movement Statistics	Statistics that are associated with the movement of material across the border of two countries.
INV_MOVE_ORDER	Material Move Order	Physical movement of inventory from one location to another within a warehouse or other facility. It does not involve a transfer of the inventory between organizations.
INV_ONHAND	Inventory On Hand Balance	Inventory On Hand Balance
INV_ORGANIZATION_SETUP	Inventory Organization Setup	Inventory Organization Setup
INV_PICK_RELEASE_PUB	Inventory Pick Release	Inventory allocation in support of pick release
INV_POSITION	Inventory Position	It indicates on-hand balance of an Inventory Organization Hierarchy for a particular time bucket including quantity received, quantity issued and ending balance.

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
INV_REPLENISHMENT	Inventory Replenishment	Inventory Material Replenishment
INV_RESERVATION	Material Reservation	Inventory Material Reservation
INV_SALES_ORDERS	Inventory Sales Order	It indicates inventory sales order tracking with references to the order in Oracle Order Management or a third party order management system.
INV_SERIAL_NUMBER	Inventory Serial Number	Inventory Serial Number
INV_SUPPLIER_CONSIGNED_INV	Supplier Consigned Inventory	Goods that physically reside in an inventory organization but are owned by a supplier.
INV_UNIT_OF_MEASURE	Unit Of Measure	Inventory Unit Of Measure
IPM_DOCUMENT	Imaging Document	Electronic documentation to facilitate the entry and completion of transactions in the Oracle E-Business Suite.
IRC_AGENCY	Recruiting Agency	Third party agency authorized to recruit for a Vacancy.
IRC_CANDIDATE_NOTIFY_PREFS	Candidate Recruitment Notification Preferences	Candidate Recruitment Notification Preferences
IRC_CANDIDATE_SAVED_SEARCH	Candidate Recruitment Saved Search	Candidate Recruitment Saved Search
IRC_CANDIDATE_WORK_PREFERENCES	Candidate Recruitment Work Preferences	Candidate Recruitment Work Preferences
IRC_DEFAULT_JOB_POSTING	Default Job Posting	Default Job Posting
IRC_JOB_BASKET	Job Basket	Job Basket

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
IRC_JOB_OFFER	Job Offer	It contains details of a job to be offered to a Recruitment Candidate.
IRC_JOB_OFFER_LETTER_TEMPLATE	Job Offer Letter Template	It is a template for a Job Offer letter.
IRC_JOB_OFFER_NOTES	Job Offer Note	Notes for a Job Offer
IRC_JOB_POSTING	Job Posting	Job Posting
IRC_JOB_SEARCH_LOCATION	Job Search Location	It contains locations for the Candidate Recruitment Saved Search or for the Candidate Recruitment Work Preferences.
IRC_JOB_SEARCH_PROF_AREA	Job Search Professional Area	It contains Professional Areas for the Candidate Recruitment Saved Search or for the Candidate Recruitment Work Preferences.
IRC_NOTIFICATION	iRecruitment Notification	Notifications that are sent to recruiter, interviewer and candidate.
IRC_RECRUITING_3RD_PARTY_SITE	Recruiting Third Party Site	Recruiting Third Party Site
IRC_RECRUITING_DOCUMENT	Recruiting Document	Recruiting Document
IRC_RECRUITING_SITE	Recruiting Site	Recruiting Site
IRC_RECRUITING_TEAM	Recruiting Team	Recruiting Team
IRC_RECRUITMENT_CANDIDATE	Recruitment Candidate	Recruitment Candidate

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
IRC_VACANCY_CONSIDERATION	Vacancy Consideration	Vacancy Consideration
JE_ES_WHT	Spanish Withholding Tax Transaction	Spanish Withholding Tax Transaction stores withholding tax transactions from Payables and other external sources.
JL_BR_AP_BANK_COLLECTION_DOC	Brazilian Payables Bank Collection Document	Brazilian Payables Bank Collection Document
JL_BR_AR_BANK_RETURN_DOC	Brazilian Receivables Bank Return Document	Brazilian Receivables Bank Return Document
JTA_BUSINESS_RULE	Business Rule	Business Rule for Escalation or Auto Notifications
JTA_ESCALATION	Customer Escalation Management	It manages customer's escalation of some key business entities like Service Requests, Tasks, etc.
JTF_RS_ DYNAMIC_GROUP	Resource Group (Dynamic)	Dynamic Resource Group (defined using dynamic SQL statements)
JTF_RS_DYNAMIC_GROUP	Resource Dynamic Group	Dynamic Resource Group (defined using dynamic SQL statements)
JTF_RS_GROUP	Resource Group	Grouping of Individual Resources
JTF_RS_GROUP_MEMBER	Resource Group Member	Members within a Group
JTF_RS_GROUP_MEMBER_ROLE	Resource Group Member Role	Roles assigned to members in a group
JTF_RS_GROUP_RELATION	Resource Group Hierarchy	Resource Group Hierarchy Element

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
JTF_RS_GROUP_USAGE	Resource Group Usage	Functional use of Resource Groups in different applications
JTF_RS_RESOURCE	Individual Resource	Individual Resource
JTF_RS_RESOURCE_AVAILABILITY	Resource Availability	Whether an individual resource is available (Yes/No) for work assignments at present time.
JTF_RS_RESOURCE_LOV	Resource	Person and non-person resources
JTF_RS_RESOURCE_SKILL	Resource Skill	Resource Skillsets for work assignment
JTF_RS_RESOURCE_SKILL_LEVEL	Resource Skill Level	Skill Levels indicating Novice, Expert, Intermediate for different skills
JTF_RS_ROLE	Person Resource Role	Roles assigned to an individual resource
JTF_RS_ROLE_RELATION	Person Resource Role Hierarchy	Hierarchy of Roles associated with individual resources, groups, and group members.
JTF_RS_SALESREP	Sales Representative	Individual Resources that represent Enterprise Sales Force.
JTF_RS_SALES_GROUP_HIERARCHY	Sales Group Hierarchy	Sales Group Hierarchy
JTF_RS_SRP_TERRITORY	Sales Representative Territory	Territories that are assigned to Sales people. **Note:** These are not to be confused with Territory Manager.

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
JTF_RS_TEAM	Resource Team	Teams represent collection of people, and groups.
JTF_RS_TEAM_MEMBER	Resource Team Member	Members of a team that includes individuals, as well as groups.
JTF_RS_TEAM_USAGE	Resource Team Usage	It represents functional use of Resource Teams in different applications.
JTF_RS_UPDATABLE_ATTRIBUTE	Updatable Attributes for Resources	Individual resource information that is allowed to be modified.
JTF_RS_WF_EVENT	Resource Business Event	Actions in Resource Manager that raise Workflow Business Events.
JTF_RS_WF_ROLE	Resource Workflow Role	It is the Workflow Role representing Individual, Group, and Team resources.
JTF_RS_WF_USER_ROLE	Resource Workflow User Role	It is the Workflow User Role representing resource roles, group members, or team members.
JTH_INTERACTION	Customer Interaction	It is the communication or attempted communication with a customer party.
JTH_INTERACTION_ACTIVITY	Customer Interaction Activity	A business event that occurs during an interaction with a customer party.
JTH_INTERACTION_MEDIA	Customer Interaction Media	It contains the details of the communication used in an interaction. Call, E-mail, Web, etc.

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
JTY_TERRITORY	Territory	Territories for sales representatives, service engineers and collections agents
LSH_APPLICATIONAREA	Application Area	A collection of objects that define a business application. Objects can be Business Area, Data Mart, Loadset, Program, Report Set, Table, Workflow etc.
LSH_BUSINESSAREA	Business Area	A Business Area acts as an interface with an External Visualization System (EVS). For example, a Discoverer Business Area is an interface with the Oracle Discoverer Visualisation tool.
LSH_DATAMART	Data Mart	A Data Mart stores data exported from the Transactional System and is usually used for Analytical purposes.
LSH_DOMAIN	Life Sciences Data Hub Domain	The top level container that owns Application Areas and is used to store object definitions in the Library. The definitions can be Business Area, Data Mart, Loadset, Program, etc.
LSH_EXECUTIONSETUP	Life Sciences Data Hub Execution Setup Information	It is a defined object that is a component of each LSH executable object instance (Programs, RS, Load Sets, Workflows etc.) whose purpose is to control the execution of the executable object.

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
LSH_EXECUTION_FWK	Life Sciences Data Hub Execution Job	An entity that provides the surround and the set the rules for the execution of an object. Examples are Job Submission API, Job Log API, etc
LSH_GENERIC_OBJECT	Life Science Data Hub Generic Object	Life Science Data Hub Generic Object
LSH_LOADSET	External Data Load Run	A Load Set is an executable object that is used to define the structure and behavior of a Program-like structure that is used for loading data from an outside system.
LSH_MAPPING	Life Science Data Hub Column Mapping	A mapping defines a column level mapping between a Table like object and a View like object.
LSH_OBJ_CLASSIFICATION	Object Classification	An entity that provides the categories and rules for classifying an object.
LSH_OBJ_SECURITY	Life Sciences Data Hub Object Security Policy	An entity that provides a set of rules to define and implement data security on objects.
LSH_OUTPUT	Life Sciences Data Hub Output	This is the actual Output that is generated on execution of an executable object, such as a Report Set output, a Data Mart output, a Program Output.
LSH_PARAMETER	API Parameter	A defined object that acts as a simple scalar variable and is based on a variable, such as an input/output Parameter of a Program, Report Set, etc.

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
LSH_PARAMETERSET	Parameter Set	A collection of interrelated Parameters
LSH_PLANNEDOUTPUT	Life Science Data Hub Planned Output	A Planned Output is an expected output when an LSH object is executed. It is defined with the executable object.
LSH_PROGRAM	Life Sciences Data Hub Program	A Program is a metadata object that is used to define the structure and behavior of a Program-like structure that is used for processing and/or reporting on set of data.
LSH_REPORTSET	Report Set	A Report Set is a group of reports used to define the structure and behavior of a hierarchical structure that is intended for simultaneously reporting on sets of data.
LSH_SOURCECODE	Software Source Code	A Source Code is the actual program code which is executed when a Program is run.
LSH_TABLE	Metadata Registered Data Object	It is a metadata description of a table-like object (for example a Oracle view or a SAS dataset).
LSH_UTILITY	Life Sciences Data Hub Setup Utility	It is a set of tools and utilities.
LSH_VALIDATION	Life Sciences Data Hub Validation	An entity that provides the rules for validating an object in the application.

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
LSH_VARIABLE	Life Science Data Hub Variable	A LSH defined object equivalent to a SAS variable or Oracle table column that serves as a source definition for LSH Parameters and Table Columns.
LSH_WORKAREA	Application Work Area	A container within an Application Area that provides the definer a place to prepare related LSH Definitional objects for release and installation to a LSH schema.
MES_COMPLETION_TRANSACTION	Assembly Completion in MES	Business Entity for Assembly Completion in MES
MES_MATERIAL_TRANSACTION	MES Material Transaction	Business Entity for Material Transaction in MES
MES_MOVE_TRANSACTION	MES Move Transaction	Business Entity for Move Transaction in MES
MES_TIME_ENTRY	Time Entry in MES	Import Time Entry Record in Discrete Manufacturing Execution system
MSC_ATP_ENQUIRY	ATP Enquiry	This interface checks the availability for the item(s) and returns their availability picture.
MSC_FORECAST	Supply Chain Forecast	This interface creates a forecast for supply chain planning.
MSC_NOTIFY_PLAN_OUTPUT	Supply Chain Planned Order	Supply chain planned order
MSC_ON_HAND	Supply Chain Plan On Hand Inventory	This interface creates on hand supply records for supply chain planning.

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
MSC_PLANNING_SUPPLY_DEMAND	Collaborative Planning Supply / Demand	This interface is used to create any supply / demand records in Collaborative Planning.
MSC_PURCHASE_ORDER	Supply Chain Plan Purchase Order	This interface creates a purchase order supply for supply chain planning.
MSC_REQUISITION	Supply Chain Plan Requisition	A Purchase Requisition for Supply Chain Planning
MSC_SALES_ORDER	Supply Chain Plan Sales Order	This interface creates a sales order demand for supply chain planning.
MSC_SHIPMENT_NOTICE	Supply Chain Plan Advanced Shipment Notice	This interface creates an inbound intransit supply for supply chain planning.
MSC_WORK_ORDER	Supply Chain Plan Work Order	This interface creates a work order supply for supply chain planning.
NETTING_BATCH	Netting Batch	Netting batch is a set of payables and receivables transactions.
OCM_GET_DATA_POINTS	Credit Review Data Point	It is a list of Data Points (Criterion items against which the credit standing of a organization is reviewed) for a given credit classification, review type, data point category, or subcategory.
OCM_GET_EXTRL_DECSN_PUB	Imported Credit Score and Recommendation	Import score and recommendations from external source
OCM_GUARANTOR_CREDIT_REQUEST	Guarantor Credit Request	It allows user to create Guarantor credit request.

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
OCM_RECOMMENDATIONS	Credit Recommendation	It is the recommendation/decision made by reviewer of the credit request. For example, a standard recommendation is "Approve/Reject".
OCM_WITHDRAW_CREDIT_REQUEST	Credit Request Withdrawal	It allows user to withdraw a credit request.
OIE_CREDIT_CARD_TRXN	Credit Card Transaction	Credit Card Transaction
OIE_PCARD_TRXN	Procurement Card Transaction	Procurement Card Transaction
OIR_REGISTRATION	Self Registration of user	Self Registration of external user of the application
OKC_DELIVERABLE	Contract Deliverable	Contract Deliverable
OKC_LIBRARY_ARTICLE	Contract Library Article	Contract Library Article
OKC_LIBRARY_CLAUSE	Contract Library Clause	Contract library clause
OKC_REPOSITORY_CONTRACT	Repository Contract	A contract that handles outside the normal purchasing or sales flows, such as a non-disclosure agreement or a partnership agreement. These contracts are stored in the Contract Repository.
OKC_REP_CONTRACT	Repository Contract(1)	A contract that handles outside the normal purchasing or sales flows, such as a non-disclosure agreement or partnership agreement. These contracts are stored in the Contract Repository(1).
OKE_CONTRACT	Project Contract	Project Contract

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
OKL_ACCOUNT_DISTRIBUTION	Lease Account Distribution	Lease Account Distribution
OKL_ACCOUNT_ID	Lease Account	Lease Account
OKL_AGREEMENT	Lease Agreement	Lease Agreement
OKL_ASSET_MANAGEMENT	Asset Management	Manage portfolios and asset returns
OKL_COLLECTION	Collection	Bill, collect cash and manage collections from customers
OKL_COLLECTION_CASE	Lease Collection Case	Lease Collection Case
OKL_CONTRACT	Lease Contract	Lease Contract
OKL_CONTRACT_LIFECYCLE	Contract Management Lifecycle	It manages revisions, termination and renewals of contracts.
OKL_CONTRACT_PARTY	Lease Contract Party	Lease Contract Party
OKL_CONTRACT_PAYMENT	Lease Contract Payment	Lease Contract Payment
OKL_CONTRACT_TERM	Lease Contract Term	Lease Contract Term
OKL_DISBURSEMENT	Disbursement	Process manually initiated or automated disbursements
OKL_EXECUTE_FORMULA	Lease Formula	Lease Formula
OKL_FINANCIAL_PRODUCT	Lease Contract Financial Product	Financial Product specified in lease contract
OKL_INSURANCE	Lease Insurance	Lease Insurance
OKL_INTEREST	Lease Interest	Lease Interest

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
OKL_INVESTMENT_PROGRAM	Manage Investment Program	It manages investor accounts and investment agreements.
OKL_LATE_POLICY	Lease Late Payment Policy	Lease Late Payment Policy
OKL_LEASE_RATE	Lease Rate Set	Lease Rate Set
OKL_MARKETING_PROGRAM	Marketing Program	It manages internal and partner pricing programs.
OKL_ORIGINATION	Origination	It manages primary agreements, author lease and loan contracts.
OKL_REMARKETING	Remarketing	It manages sale of assets to vendors and third parties.
OKL_RESIDUAL_VALUE	Lease Residual Value	Lease Residual Value
OKL_RISK_MANAGEMENT	Risk Management	It manages credit, pricing, approval and insurance policies.
OKL_SALES	Sales	Qualify, quote and manage deal opportunities
OKL_STREAM	Lease Stream	Lease Stream
OKL_TERMINATION_QUOTE	Lease Termination Quote	Lease Termination Quote
OKL_THIRD_PARTY_BILLING	Lease Third Party Billing	Lease Third Party Billing
OKL_UNDERWRITING	Manage Underwriting	It manages credit applications and lines.
OKL_VENDOR_RELATIONSHIP	Manage Vendor Relationship	It manages vendor accounts and agreements.

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
OKS_AVAILABLE_SERVICE	Service Availability	APIs for retrieving customer service information, specifically, duration of a service, availability of service for a customer and list of services which can be ordered for a customer.
OKS_CONTRACT	Service Contract	Service Contract
OKS_COVERAGE	Service Contract Coverage	Service contract coverage service terms
OKS_ENTITLEMENT	Service Contract Entitlement	Service contract customer entitled services
OKS_IMPORT	Service Contracts Import	Service contract import is a process of importing the historical or ongoing contracts data from an external or a legacy system into the Oracle service contract tables.
ONT_SALES_AGREEMENT	Sales Agreement	This is a business document that outlines the agreement between a Customer and Supplier committing to order and deliver a specified amount or quantity over an agreed period of time.
ONT_SALES_ORDER	Sales Order	Sales Order is a business document containing customer sales order information. This entity is used by several Oracle E-Business Suite applications.
OTA_CATALOG_CATEGORY	Learning Catalog Category	Learning Catalog Category

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
OTA_CERTIFICATION	Learning Certification	Catalog object that offers learners the opportunity to subscribe to and complete one time and renewable certifications.
OTA_CHAT	Learning Chat	Scheduled live discussion that enables learners and instructors to exchange messages online.
OTA_CONFERENCE_SERVER	Conference Server	Conference server integrates OLM with Oracle Web Conferencing (OWC) to deliver online synchronous classes.
OTA_COURSE_PREREQUISITE	Course Prerequisite	A course or competency that a learner must or should complete before enrolling in a given class.
OTA_ENROLLMENT_JUSTIFICATION	Learning Enrollment Justification	Each enrollment justification and its associated priority level can determine the order by which enrollees are automatically placed in a class.
OTA_ENROLLMENT_STATUS_TYPE	Learning Enrollment Status Type	It indicates predefined enrollment statuses (Requested, Placed, Attended, Waitlisted, Cancelled).
OTA_FINANCE_HEADER	Learning Finance Header	It is a record of a monetary amount against a class, a learner enrollment, or a resource booking.
OTA_FINANCE_LINE	Learning Finance Line	It is an individual financial transaction within a finance header.

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
OTA_FORUM	Learning Forum	It represents message board that learners and instructors use to post general learning topics for discussion.
OTA_LEARNER_ENROLLMENT	Learner Enrollment	Learner Enrollment
OTA_LEARNING_ANNOUNCEMENT	Learning Announcement	Learning Announcement
OTA_LEARNING_CATALOG_CAT_USE	Learning Catalog Category Usage	Learning Catalog Category Usage
OTA_LEARNING_CLASS	Learning Class	Learning Class
OTA_LEARNING_COURSE	Learning Course	Learning Course
OTA_LEARNING_CROSS_CHARGE	Learning Cross Charge Setup	Learning Cross Charge Setup
OTA_LEARNING_EXTERNAL	Learning External Record	A class or course that a person has attended, not scheduled in the internal learning catalog.
OTA_LEARNING_OFFERING	Learning Offering	Learning Offering
OTA_LEARNING_OFFER_RES_CHKLST	Learning Offering Resource Checklist	Learning Offering Resource Checklist
OTA_LEARNING_PATH	Learning Path	Learning Path
OTA_LEARNING_PATH_CATEGORY	Learning Path Category	Learning Path Category
OTA_LEARNING_PATH_COMPONENT	Learning Path Component	Learning Path Component

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
OTA_LP_SUBSCRIPTION	Learning Path Subscription	It contains subscriptions for all Learning Paths and Components. For Example, Subscriptions to Catalog Learning Paths and Learning Paths created by Managers from Appraisals, Suitability Matching etc.
OTA_RESOURCE	Learning Resource	It is a person or an object needed to deliver a class, such as a named instructor or a specific classroom.
OTA_RESOURCE_BOOKING	Learning Resource Booking	Learning Resource Booking
OTA_TRAINING_PLAN	Training Plan	Training Plan
OZF_ACCOUNT_PLAN	Trade Account Plan	Account Plan for Trade Planning and Promotion activities
OZF_BUDGET	Sales and Marketing Budget	It is the budget for Promotional Offer, Marketing Campaigns, Events, and other marketing activities.
OZF_CLAIM	Trade Claim	Claims that customers could be seeking money against, such as Promotional claims, breakages, transportation errors etc.
OZF_EXTRACT	Accounting Extract	Accounting extract for a third party General Ledger. It includes details on account, amount, customer, product for accrual and adjustment, as well as promotional claim settlements.
OZF_INDIRECT_SALES	Indirect Sales	Point of Sales Data, chargebacks etc,

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
OZF_OFFERS	Promotional Offer	Promotional Offers or Discounts that are given to Customers from a Vendors Sales or Marketing Organization.
OZF_QUOTA	Trade Planning Quota	Quota and Targets for Trade Planning and Promotional Activities
OZF_SOFT_FUND	Partner Fund	Partner Fund Requests
OZF_SPECIAL_PRICING	Special Pricing	Special Pricing Requests
OZF_SSD_BATCH	Supplier Ship and Debit Batch	Supplier ship and debit batch is essentially a claim that the distributor submits to the supplier for approval and payment. The batch contains accruals for which the supplier is expected to make payment for.
OZF_SSD_REQUEST	Supplier Ship and Debit Request	An agreement that allows distributor to request a special price from supplier. The ship and debit request will allow specifications with respect to the quantity limits associated to the price reduction, product and period.
PAY_BALANCE	Payroll Balance	Payroll Balance
PAY_BALANCE_ADJUSTMENT	Payroll Balance Adjustment	Payroll Balance Adjustment
PAY_BATCH_ELEMENT_ENTRY	HRMS Batch Element Entry	HRMS Batch Element Entry
PAY_CONTRIBUTION_USAGE	Payroll Contribution Usage	Payroll Contribution Usage

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
PAY_COST_ALLOCATION	Payroll Cost Allocation	Payroll Cost Allocation
PAY_DEFINED_BALANCE	Payroll Defined Balance	Payroll Defined Balance
PAY_ELEMENT	HRMS Element	HRMS Element
PAY_ELEMENT_CLASSIFICATION	HRMS Element Classification	It describes categories of HRMS Elements such as Earnings, Deductions and Information. These influence subsequent processing.
PAY_ELEMENT_ENTRY	HRMS Element Entry	HRMS Element Entry
PAY_ELEMENT_LINK	HRMS Element Eligibility Criteria	HRMS Element Eligibility Criteria
PAY_EMP_TAX_INFO	Employee Tax Information	Employee Tax Information
PAY_FORMULA_RESULT	Payroll Processing Result Rule	It indicates how the Formula is to be processed and how its result is to be used by the Payroll processes.
PAY_ITERATIVE_RULE	Payroll Iterative Rule	Payroll Iterative Rule
PAY_LEAVE_LIABILITY	Leave Liability	It describes leave type and associated definitions utilized by the Leave Liability process.
PAY_ORG_PAYMENT_METHOD	Organization Payment Method	It is a Payroll Payment Method used by the Organization for employee compensation.
PAY_PAYMENT_ARCHIVE	Payroll Payment Archive	Payroll Payment Archive
PAY_PAYROLL_DEFINITION	Payroll Definition	Payroll Definition
PAY_PAYROLL_EVENT_GROUP	Payroll Event Interpretation Group	Payroll Event Interpretation Group

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
PAY_PAYROLL_TABLE_REC_EVENT	Payroll Table Recordable Event	Payroll Table Recordable Event
PAY_PERSONAL_PAY_METHOD	Personal Payment Method	Personal Payment Method
PAY_PROVINCIAL_MEDICAL	Provincial Medical Account	Provincial Medical Account
PAY_RUN_TYPE	Payroll Run Type	Payroll Run Type
PAY_TIME_DEFINITION	Payroll Time	This holds information about period of time and its usage.
PAY_USER_DEFINED_TABLE	HRMS User Defined Table	HRMS User Defined Table
PAY_WORKERS_COMPENSATION	Workers Compensation	Workers Compensation
PA_AGREEMENT	Project Customer Agreement	Project Customer Agreement
PA_BILLING_EVENT	Project Billing Event	Project Billing Event
PA_BUDGET	Project Budget	Project Budget
PA_BURDEN_COST	Project Burden Cost	Project Burden Cost
PA_CAPITAL_ASSET	Project Capital Asset	Project Capital Asset
PA_CUSTOMER_INVOICE	Project Customer Invoice	Project Customer Invoice
PA_EXPENDITURE	Project Expenditure	Project Expenditure
PA_EXPENSE_RPT_COST	Project Expense Report Cost	Project Expense Report Cost
PA_FINANCIAL_TASK	Project Financial Task	Project Financial Task
PA_FORECAST	Project Forecast	Project Forecast
PA_IC_TRANSACTION	Project Cross Charge	Project Cross Charge

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
PA_INTERCOMPANY_INVOICE	Project Intercompany Invoice	Project Intercompany Invoice
PA_INTERPROJECT_INVOICE	Project Interproject Invoice	Project Interproject Invoice
PA_INVENTORY_COST	Project Inventory Cost	Project Inventory Cost
PA_INVOICE	Project Invoice	Project Invoice
PA_LABOR_COST	Project Labor Cost	Project Labor Cost
PA_MISCELLANEOUS_COST	Project Miscellaneous Cost	Project Miscellaneous Cost
PA_PAYABLE_INV_COST	Project Supplier Cost	Project Supplier Cost
PA_PERF_REPORTING	Project Reporting	Project Reporting
PA_PROJECT	Project	Project
PA_PROJ_COST	Project Cost	Project Cost
PA_PROJ_DELIVERABLE	Project Deliverable	Project Deliverable
PA_PROJ_FUNDING	Project Funding	Project Funding
PA_PROJ_PLANNING_RESOURCE	Project Planning Resource	Project Planning Resource
PA_PROJ_RESOURCE	Project Resource	Project Resource
PA_RES_BRK_DWN_STRUCT	Project Resource Breakdown Structure	Project Resource Breakdown Structure
PA_REVENUE	Project Revenue	Project Revenue
PA_TASK	Project Task	Project Task
PA_TASK_RESOURCE	Project Task Resource	Project Task Resource

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
PA_TOT_BURDENED_COST	Project Total Burdened Cost	Project Total Burdened Cost
PA_USAGE_COST	Project Asset Usage Cost	Project Asset Usage Cost
PA_WIP_COST	Project Work in Process Cost	Project Work in Process Cost
PA_WORKPLAN_TASK	Project Workplan Task	Project Workplan Task
PER_APPLICANT	Applicant	Applicant
PER_APPLICANT_ASG	Applicant Assignment	Applicant Assignment
PER_APPRAISAL	Worker Appraisal	Worker Appraisal
PER_APPRAISAL_PERIOD	Appraisal Period	It defines appraisal period information to be used within a performance plan.
PER_ASSESSMENT	Worker Assessment	Worker Assessment
PER_BF_BALANCE	Third Party Payroll Balance	Third Party Payroll Balance
PER_BF_PAYROLL_RESULTS	Third Party Payroll Results	Third Party Payroll Results
PER_CHECKLIST	Person Task Checklist	It is a checklist containing tasks which can be copied and the copy assigned to an Employee, Contingent Worker or Applicant, e.g. 'New Hire Checklist'.
PER_COLLECTIVE_AGREEMENT	Collective Agreement	Collective Agreement
PER_COLLECTIVE_AGREEMENT_ITEM	Collective Agreement Item	Collective Agreement Item
PER_COMPETENCE	Competence	Competence

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
PER_COMPETENCE_ELEMENT	Competence Element	Competence Element
PER_COMPETENCE_RATING_SCALE	Competence Rating Scale	Competence Rating Scale
PER_CONFIG_WORKBENCH	HCM Configuration Workbench	It manages enterprise structure configuration workbench wizard for setting up entities such as Locations, Business Groups, Jobs and Positions.
PER_CONTACT_RELATIONSHIP	Contact Relationship	Contact Relationship
PER_CWK	Contingent Worker	Contingent Worker
PER_CWK_ASG	Contingent Worker Assignment	Contingent Worker Assignment
PER_CWK_RATE	Contingent Worker Assignment Rate	Contingent Worker Assignment Rate
PER_DISABILITY	Disability	Disability
PER_DOCUMENTS_OF_RECORD	Documents of Record	Documents of Record for an Employee, Contingent Worker, Applicant or Contact
PER_EMPLOYEE	Employee	Employee
PER_EMPLOYEE_ABSENCE	Employee Absence	Employee Absence
PER_EMPLOYEE_ASG	Employee Assignment	Employee Assignment
PER_EMPLOYMENT_CONTRACT	Employment Contract	Employment Contract
PER_ESTAB_ATTENDANCES	Schools and Colleges Attended	Schools and Colleges Attended

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
PER_EX-EMPLOYEE	Ex-Employee	Ex-Employee
PER_GENERIC_HIERARCHY	Generic Hierarchy	Generic Hierarchy
PER_GRADE	Employee Grade	Employee Grade
PER_JOB	Job	Job
PER_JOB_GROUP	Job Group	Job Group
PER_MEDICAL_ASSESSMENT	Medical Assessment	Medical Assessment
PER_OBJECTIVE_LIBRARY	Objectives Library	A repository of reusable objectives that can be either created individually or imported from an external source.
PER_ORGANIZATION_HIERARCHY	Organization Hierarchy	Organization Hierarchy
PER_PERFORMANCE_REVIEW	Employee Performance Review	Employee Performance Review
PER_PERF_MGMT_PLAN	Performance Management Plan	It indicates the parameters of the performance management process, including the performance period, population and appraisal periods.
PER_PERSON	HR Person	HR Person
PER_PERSONAL_CONTACT	Personal Contact	Personal Contact
PER_PERSONAL_SCORECARD	Person Scorecard	One worker's objectives for a performance management plan, which provides a goal setting, performance review and scoring basis.

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
PER_PERSON_ADDRESS	Person Address	Person Address
PER_PHONE	Phone	Phone
PER_POSITION	Position	Position
PER_POSITION_HIERARCHY	Position Hierarchy	Position Hierarchy
PER_PREVIOUS_EMPLOYMENT	Previous Employment	Previous Employment
PER_QUALIFICATION	Person Qualification	Person Qualification
PER_RECRUITMENT_ACTIVITY	Recruitment Activity	Recruitment Activity
PER_SALARY_PROPOSAL	Salary Proposal	Salary Proposal
PER_SALARY_SURVEY	Salary Survey	Salary Survey
PER_SCORECARD_SHARING	Scorecard Access	It holds the list of persons and access permissions for a scorecard for which the owner of the scorecard has granted access.
PER_SECURITY_PROFILE	Security Profile	Security Profile
PER_SUPPLEMENTARY_ROLE	HR Supplementary Role	HR Supplementary Role
PER_VACANCY	Vacancy	Vacancy
PER_VACANCY_REQUISITION	Vacancy Requisition	Vacancy Requisition
PER_WORK_COUNCIL_ELECTION	Work Council Election	Work Council Election
PER_WORK_INCIDENT	Work Incident	Work Incident

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
PN_CUSTOMER_SPACE_ASSIGNMENT	Customer Space Assignment	Customer Space Assignment
PN_EMPLOYEE_SPACE_ASSGNMENT	Employee Space Assignment	Employee Space Assignment
PN_PROPERTY	Space	A property or component of property, such as a building, land parcel, floor, or office.
PN_RECOVERABLE_EXPENSE	Property Recoverable Expense	Property Recoverable Expense
PN_VOLUME_HISTORY	Variable Rent Volume History	Variable Rent Volume History
PJM_INVENTORY	Project Manufacturing Inventory	Inventory tracked by project, when dealing with permanent and temporary transfers from one project to another, or from common inventory to project inventory.
PO_ACKNOWLEDGEMENT	Purchase Order Acknowledgement	Purchase Order Acknowledgement
PO_ADVANCED_SHIP_NOTFN	Advanced Shipment Notification	Advanced Shipment Notification
PO_APPROVAL	Purchase Order Approval	Purchase Order Approval
PO_APPROVAL_HIERARCHY	Purchase Order Approval Hierarchy	Purchase Order Approval Hierarchy
PO_APPROVED_SUPPLIER_LIST	Approved Supplier List	Approved Supplier List
PO_ATTACHMENTS	Procurement Attachments	Procurement Attachments
PO_AUCTION	Auction	Auction
PO_AWARD	Award	Award

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
PO_BIDDING_ATTRIBUTES	Bidding Attributes	Bidding Attributes
PO_BLANKET_PURCHASE_AGREEMENT	Blanket Purchase Agreement	Blanket Purchase Agreement
PO_BLANKET_RELEASE	Purchasing Blanket Release	A blanket release is issued against a blanket purchase agreement to place the actual order.
PO_CLM_AWARD	CLM Award	A Contract Lifecycle Management Award document outlines the agreement between the government and a supplier to provide goods and services.
PO_CLM_IDV	CLM Indefinite Delivery Vehicle	A Contract Lifecycle Management Indefinite Delivery Vehicle identifies any undefined strategic requirement for goods and services over a specified period of time.
PO_CATALOG	Purchasing Catalog	Purchasing Catalog
PO_CATALOG_CATEGORY	Purchasing Catalog Category	Purchasing Catalog Category
PO_CHANGE	Purchase Order Change	Purchase Order Change
PO_CONSUMPTION_ADVICE	Consigned Inventory Consumption Advice	Release or Standard PO for Consigned Consumption
PO_CONTRACT	Purchasing Contract	Purchasing Contract
PO_CONTRACT_PURCHASE_AGREEMENT	Contract Purchase Agreement	Contract Purchase Agreement
PO_CONTRACT_TEMPLATE	Purchasing Contract Template	Purchasing Contract Template

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
PO_CONTRACT_TERM	Purchasing Contract Term	Purchasing Contract Term (Articles, Deliverables and Contract Documents)
PO_DOCUMENT_APPROVER	Purchasing Document Approver	Purchasing Document Approver
PO_EXPENSE_RECEIPT	Expense Receipt	Expense Receipt
PO_GLOBAL_BLANKET_AGREEMENT	Global Blanket Purchase Agreement	Global Blanket Purchase Agreement
PO_GLOBAL_CONTRACT_AGREEMENT	Global Contract Purchase Agreement	Global Contract Purchase Agreement
PO_GOODS_RECEIPT	Goods Receipt	Goods Receipt
PO_GOODS_RETURN	Goods Return	Goods Return
PO_INTERNAL_REQUISITION	Internal Requisition	Internal Requisition
PO_NEGOTIATION	Sourcing Negotiation	Sourcing Negotiation
PO_PLANNED_PURCHASE_ORDER	Planned Purchase Order	Planned Purchase Order
PO_PLANNED_RELEASE	Planned PO Release	Planned PO Release
PO_PRICE_BREAKS	Sourcing Price Break	Sourcing Price Break
PO_PRICE_DIFFERENTIAL	Purchasing Price Differential	Purchasing Price Differential holds the price differentials for the rate based lines for requisition lines, PO lines or Blanket pricebreaks based on the entity type.
PO_PRICE_ELEMENTS	Sourcing Price Element	Sourcing Price Element
PO_PURCHASE_REQUISITION	Purchase Requisition	Purchase Requisition

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
PO_QUOTE	Sourcing Quote	Sourcing Quote
PO_RECEIPT_CORRECTION	Receipt Correction	Receipt Correction
PO_RECEIPT_TRAVELER	Receipt Traveler	Receipt Traveler
PO_REQUISITION_APPROVAL	Requisition Approval	Requisition Approval
PO_REQ_APPROVAL_HIERARCHY	Requisition Approval Hierarchy	Requisition Approval Hierarchy
PO_RFI	Request for Information	Request for Information
PO_RFQ	Request for Quotation	Request for Quotation
PO_RFQ_RESPONSE	RFQ Response	RFQ Response
PO_SERVICES_RECEIPTS	Services Receipt	Services Receipt
PO_SHIPMENT_AND_BILLING_NOTICE	Shipment / Billing Notice	Shipment / Billing Notice
PO_SOURCING_BID	Sourcing Bid	Sourcing Bid
PO_SOURCING_RULES	Sourcing Rule	Sourcing Rule
PO_SOURCING_RULE_ASSIGNMENTS	Sourcing Rule Assignment	Sourcing Rule Assignment
PO_STANDARD_PURCHASE_ORDER	Standard Purchase Order	Standard Purchase Order
PO_SUPPLIER_BANK_ACCOUNT	Supplier Bank Account	Supplier Bank Account
PQH_ADDITIONAL_SECOND_PENSION	Additional Second Pension	Additional Second Pension
PQH_DEFAULT_HR_BUDGET_SET	Default HR Budget Set	Default HR Budget Set

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
PQH_EMEA_SENIORITY_SITUATION	European Seniority Situation	European Seniority Situation
PQH_EMPLOYEE_ACCOMMODATION	Employee Accommodation	Employee Accommodation
PQH_EMPLOYER_ACCOMMODATION	Employer Provided Accommodation	Employer Provided Accommodation
PQH_FR_CORPS	French CORPS	French CORPS
PQH_FR_SERVICES_VALIDATION	French Services Validation	French Services Validation
PQH_FR_STATUTORY_SITUATION	French Statutory Situation	French Statutory Situation
PQH_GLOBAL_PAY_SCALE	Global Pay Scale	Global Pay Scale
PQH_POS_CTRL_BUSINESS_RULE	Position Control Business Rule	Position Control Business Rule
PQH_POS_CTRL_ROUTING	Position Control Routing	Position Control Routing
PQH_POS_CTRL_TRANS_TEMPLATE	Position Control Transaction Template	Position Control Transaction Template
PQH_RBC_RATE_MATRIX	Person Eligibility Criteria Rates Matrix	Rate Matrix stores different criteria value combinations and the rate a person is eligible for if the person's value matches the criteria values.
PQH_REMUNERATION_REGULATION	Remuneration Regulation	Remuneration Regulation
PQH_WORKPLACE_VALIDATION	Workplace Validation Process	Workplace Validation
PQP_PENSION_AND_SAVING_TYPE	Pension and Saving Type	Pension and Saving Type

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
PQP_VEHICLE_ALLOCATION	Vehicle Allocation	Vehicle Allocation
PQP_VEHICLE_REPOSITORY	Vehicle Repository	Vehicle Repository
PRP_PROPOSAL	Sales Proposal	Sales Proposal
PSP_EFF_REPORT_DETAILS	Employee Effort Report	It summarizes employee's labor distributions over a period of time. It is used to ensure accurate disbursement of labor charges to comply with Office of Management and Budget Guidelines.
PV_OPPORTUNITY	Partner Opportunity Assignment	It supports the assignment of indirect opportunities to partners.
PV_PARTNER_PROFILE	Partner Profiling	It is the extensible attribute model used to capture additional information about a partner and their contacts.
PV_PROGRAM	Partner Program Management	It represents the partner program management framework which includes the creation/maintenance of partner programs and the associated partner enrollments/memberships into those programs.
PV_REFERRAL	Partner Business Referral	Partner creates referrals to refer business to vendor. If referral results in a sale, the partner gets compensated. Partner register deals with vendor for non-competition purposes.
QA_PLAN	Quality Collection Plan	Quality Collection Plan

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
QA_RESULT	Quality Result	It indicates collection plan result data collected directly, through transactions or collection import.
QA_SPEC	Quality Specification	A requirement for a characteristic for an item or item category specific to a customer or supplier.
QOT_QUOTE	Sales Quote	Sales Quote
QP_PRICE_FORMULA	Price Formula	Price Formula
QP_PRICE_LIST	Price List	Price List
QP_PRICE_MODIFIER	Price Modifier	Price Modifier
QP_PRICE_QUALIFIER	Price Qualifier	Price Qualifier
REPAIR_ORDER	Repair Order	Repair Order
RLM_CUM	Supplier Shipment Accumulation	It is used to track the total shipments made by the supplier for a particular customer item, based on CUM management setup.
RLM_SCHEDULE	Customer Demand Schedule	It refers to customers production material release.
RRS_SITE	Site	It indicates the spatial location of an actual or planned structure or set of structures (as a building, business park, communication tower, highway or monument).
SOA_DIAGNOSTICS	Diagnostics for Oracle E-Business Suite Integrated SOA Gateway	Diagnostics for Oracle E-Business Suite integrated SOA Gateway

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
UMX_ACCT_REG_REQUESTS	User Account Request	It represents requests made for user accounts, needed to gain system access.
UMX_ROLE	Security Role	It represents a set of permissions in the security system. Roles are assigned to users and can be defined in role inheritance hierarchies. A Responsibility is a special type of role.
UMX_ROLE_REG_REQUESTS	Security Role Request	It represents requests made for roles (as defined in the security system) to gain access to a secured part of the system.
WF_APPROVALS_SERVICES	Approvals Data Services	It provides services that can be invoked by a client application to retrieve Oracle Workflow approvals summary details and perform approval actions.
WF_ENGINE	Workflow Item	It indicates a workflow Item including processes, functions, notifications and event activities.
WF_EVENT	Business Event	Business Event
WF_NOTIFICATION	Workflow Notification	Workflow Notification
WF_USER	Workflow Directory User	Workflow Directory User
WF_WORKLIST	Workflow Worklist Content	Approve workflow entities (Expense Reports, PO Request, HR Offer, HR Vacancy)
WIP_ACCOUNTING_CLASS	WIP Accounting Class	WIP Accounting Class

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
WIP_COMPLETION_TRANSACTION	WIP Assembly Completion	Business Entity for Assembly Completion in WIP
WIP_EMPLOYEE_LABOR_RATE	WIP Employee Labor Rate	WIP Employee Labor Rate
WIP_MATERIAL_TRANSACTION	WIP Material Transaction	Business Entity for Material Transaction in WIP
WIP_MOVE_TRANSACTION	WIP Shopfloor Move	WIP Shopfloor Move
WIP_PARAMETER	Work in Process Setup	Work in Process Setup
WIP_PRODUCTION_LINE	Production Line	Production Line
WIP_REPETITIVE_SCHEDULE	Repetitive Schedule	Repetitive Schedule
WIP_RESOURCE_TRANSACTION	WIP Resource Process Flow	WIP Resource Transaction
WIP_SCHEDULE_GROUP	WIP Schedule Group	WIP Schedule Group
WIP_SHOPFLOOR_STATUS	Shopfloor Status	Shopfloor Status
WIP_WORK_ORDER	Work Order	Job/Work Order
WMS_CONTAINER	Warehouse Management License Plate	Warehouse Container and License Plate Management
WMS_DEVICE_CONFIRMATION_PUB	Dispatch Task	It contains status update for dispatch task. For example, an ASRS task, a Carousel task, a Pick to Light system task.
WMS_DEVICE_INTEGRATION	Warehouse Device	Warehouse Device
WMS_EPC_PUB	Electronic Product Code	It stores Electronic Product Codes such as GTIN, GID, SSCC, etc.

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
WMS_INSTALL	Warehouse Management System Installation Check	This API has two purposes: • This API checks if WMS product is installed in the system, without which some flags are hidden on forms. • The API also returns if an organization is wms enabled.
WMS_LABEL	Label Printing	It holds information to support the printing of shipping, package, container, item and serial labels.
WMS_LICENSE_PLATE	License Plate	It is an identifier of a container instance used by shipping, warehouse management and shop floor management.
WMS_RFID_DEVICE	Warehouse Management Radio Frequency Identification	Warehouse Management Radio Frequency Identification Integration
WMS_SHIPPING_TRANSACTION	Warehouse Management Shipping Transaction	Warehouse Management truck loading and shipping
WSH_CONTAINER_PUB	Container	Vessel in which goods and material are packed for shipment.
WSH_DELIVERY	Delivery	Group of Shipment Lines
WSH_DELIVERY_LINE	Delivery Line	Shipment Line

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
WSH_EXCEPTIONS_PUB	Shipping Exception	Exceptions automatically logged for Shipping Entities such as Change Quantity, Cancel Shipment in OM, etc. Exception behavior defined as "Error", "Warning" or "Information Only".
WSH_FREIGHT_COSTS_PUB	Freight Costs	The cost of transportation services for the Shipper. For example, amount Shipper will pay carrier for transportation services.
WSH_PICKING_BATCHES_PUB	Pick Release	The process of releasing delivery lines to warehouse for allocation and picking.
WSH_TRIP	Trip	It describes a planned or historical departure of shipment from a location.
WSH_TRIP_STOPS_PUB	Trip Stop	The physical location through which a Trip will pass where goods are either dropped off or picked up.
WSM_INV_LOT_TXN	Inventory Lot Transaction	Lot based Inventory Transactions
WSM_LOT_BASED_JOB	Lot Based Job	Lot Based Job / WIP Lot
WSM_LOT_MOVE_TXN	Lot Move Transaction	Lot based jobs shopfloor move transactions
WSM_WIP_LOT_TXN	WIP Lot Transaction	Lot based WIP transactions

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
XDP_SERVICE_ORDER	Service Fulfillment Order	An order for one or more services, which need to be provisioned by Service Fulfillment Manager. The provisioning of these services often involve systems outside Oracle E-Business Suite.
XLA_JOURNAL_ENTRY	Subledger Accounting Journal Entry	Subledger Accounting Journal Entry comprising of a Header, Line and Distribution
XNB_ADD_BILLSUMMARY	Bill Summary Processing	This is used for inserting, creating, or populating new Bill Summary records into Oracle E-Business Suite from external Billing systems.
XNB_ADD_GROUPSALESORDER	Billing System Sales Order Lines Group	All the Sales order lines information is generated as one XML Message and published to third party billing application.
XNB_ADD_SALESORDER	Billing System Sales Order Addition	Sales order information is generated as XML Message and published to third party billing application.
XNB_SYNC_ACCOUNT	Billing System Customer Account Synchronization	An account information is generated as XML Message and published to third party billing application.
XNB_SYNC_ITEM	Billing System Inventory Item Synchronization	Catalog information is generated as XML Message and published to third party billing application.
XTR_BANK_BALANCE	Bank Account Balance	Bank Account Balance
XTR_DEAL_DATA	Treasury Deal	Treasury Deal

BUSINESS_ENTITY_CODE	MEANING	DESCRIPTION
XTR_MARKET_DATA	Market Rate	Financial Market Rates Data
XTR_PAYMENT	XTR Payment	Treasury Payment represents the payments that are being made.
ZX_DATA_UPLOAD	Imported Tax Content	This entity code is used in all the programs of Oracle E-Business Suite Tax Content Upload Request Set.

Example: Create Customer

```
/*#
_*This interface creates a customer. It calls the
_*customer hub API that creates a 'party' to create a
_*party of type 'customer'.
_*@rep:scope public
_*@rep:product OM
_*@rep:displayname Create Customer
_*@rep:category BUSINESS_ENTITY OM_CUSTOMER
_*@rep:lifecycle active
_*@rep:compatibility S
_*/
```

Composite Service - BPEL Annotation Guidelines

This section describes what you should know about Integration Repository annotations for Composite Services - BPEL.

Annotating Composite Services - BPEL

- You should annotate BPEL projects in *.bpel files.
- Before annotating, make sure that no comments beginning with `/*#` are present. The "slash-star-pound" characters are used to set off repository annotations, and will result in either an error or undesirable behavior if used with normal comments.
- To annotate, open the .bpel file in text editor to edit the file.
- In the .bpel file, place the annotations within the comments section in beginning of the file.
- Enter meaningful description that covers the condition under which the business event is raised and the UI action that invokes the business event.
- Define product codes in FND_APPLICATION.

- Use existing business entities for your composite services - BPEL processes. For the list of existing business entities, see Business Entity Annotation Guidelines, page A-41.

- Interface name in `<BPEL_PROCESS_NAME>.bpel` should be defined as `'oracle.apps' + product_code + '.<BPEL_PROCESS_NAME>`.

- If BPEL process name is `"BPEL_PROCESS_NAME"`, then
  - A BPEL Process Jar file should be created with `<prod>_pbel_<BPEL_PROCESS_NAME>.jar`.
  - `<prod>_pbel_<BPEL_PROCESS_NAME>.jar` file should be placed under `$product_top/patch/115/jar/bpel`.
  - `<prod>_pbel_<BPEL_PROCESS_NAME>.jar` file should be unzipped under `$product_top/patch/115/jar/bpel`.
  - BPEL file for `<BPEL_PROCESS_NAME>.jar` should be present under `$product_top/patch/115/jar/bpel/<prod>_pbel_<BPEL_PROCESS_NAME>.bpel`.
  - BPEL File Name should not be changed from `<BPEL_PROCESS_NAME>.bpel`.
  - WSDL file for `<BPEL_PROCESS_NAME>` should be present under `$product_top/patch/115/jar/bpel/<prod>_pbel_<BPEL_PROCESS_NAME>.bpel`.
  - WSDL File Name should not be changed from `<BPEL_PROCESS_NAME>.wsdl`.
  - Standalone Parser should be run on annotated `$product_top/patch/115/jar/bpel/<prod>_pbel_<BPEL_PROCESS_NAME>.bpel/bpel<BPEL_PROCESS_NAME>.bpel`.
  - `$product_top/patch/115/jar/bpel/<prod>_pbel_<BPEL_PROCESS_NAME>.bpel/bpel<BPEL_PROCESS_NAME>_bpel.ildt` should be loaded into Integration Repository.

- During the execution of a standalone parser, arcs file location of *.bpel file should be `patch/115/jar/bpel`.

Annotations for Composite Services - BPEL - Syntax

The annotations for composite services - BPEL are:

```
/*#
   * This is a bpel file for creating invoice.
   * @rep:scope public
   * @rep:displayname Create Invoice
   * @rep:lifecycle active
   * @rep:product inv
   * @rep:compatibility S
   * @rep:interface oracle.apps.inv.CreateInvoice
   * @rep:category BUSINESS_ENTITY INVOICE_CREATION
   */
```

Refer to General Guidelines for Annotations, page A-1 in Integration Repository for details of element definitions.

Required Annotations

Follow the links below to view syntax and usage of each annotation.

- Must begin with description sentence(s)
- rep:displayname, page A-121
- rep:scope, page A-119
- rep:product, page A-120
- rep:category BUSINESS_ENTITY, page A-128

Optional Annotations

- link, page A-125
- see, page A-126
- rep:lifecycle, page A-123
- rep:compatibility, page A-124
- rep:ihelp, page A-127
- rep:metalink, page A-128

Template

You can use the following template when annotating composite - BPEL files:

```
.
.
.
 /*#
  * <Put your long bpel process description here
  * it can span multiple lines>
  * @rep:scope <scope>
  * @rep:displayname <display name>
  * @rep:lifecycle <lifecycle>
  * @rep:product <product or pseudoproduct short code>
  * @rep:compatibility <compatibility code>
  * @rep:interface <oracle.apps.[product_code].[bpel_process_name]>
  * @rep:category BUSINESS_ENTITY <entity name>
  */
.
.
.
```

Example

Here is an example of an annotated composite - BPEL file:

```
////////////////////////////////////////////////////////////
Oracle JDeveloper BPEL Designer

  Created: Tue Oct 30 17:10:13 IST 2007
  Author:  <username>
  Purpose: Synchronous BPEL Process
 /*#
  * This is a bpel file for creating invoice.
  * @rep:scope public
  * @rep:displayname Create Invoice
  * @rep:lifecycle active
  * @rep:product PO
  * @rep:compatibility S
  * @rep:interface oracle.apps.po.CreateInvoice
  * @rep:category BUSINESS_ENTITY INVOICE
  */

////////////////////////////////////////////////////////////

-->
<process name="CreateInvoice">
   targetNamespace="http://xmlns.oracle.com/CreateInvoice"
     xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
     xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-
process/"
         xmlns:xp20="http://www.oracle.com/XSL/Transform/java/oracle.
tip.pc.services.functions.Xpath20"
         xmlns:ns4="http://xmlns.oracle.
com/pcbpel/adapter/file/ReadPayload/"
         xmlns:ldap="http://schemas.oracle.com/xpath/extension/ldap"
         xmlns:xsd="http://www.w3.org/2001/XMLSchema"
         xmlns:ns5="http://xmlns.oracle.com/bpel/workflow/xpath"
         xmlns:client="http://xmlns.oracle.com/CreateInvoice"
         xmlns:ns6="http://xmlns.oracle.
com/bpel/services/IdentityService/xpath"
         xmlns:ora="http://schemas.oracle.com/xpath/extension"
         xmlns:ns1="http://xmlns.oracle.
com/soaprovider/plsql/AR_INVOICE_API_PUB_2108/CREATE_SINGLE_INVOICE_1037
895/"
         xmlns:ns3="http://xmlns.oracle.
com/soaprovider/plsql/AR_INVOICE_API_PUB_2108/APPS/BPEL_CREATE_SINGLE_IN
VOICE_1037895/AR_INVOICE_API_PUB-24CREATE_INV/"
         xmlns:ns2="http://xmlns.oracle.com/pcbpel/adapter/appscontext/"
         xmlns:bpelx="http://schemas.oracle.com/bpel/extension"
   xmlns:orcl="http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.
services.functions.ExtFunc">

  <!--
////////////////////////////////////////////////////////////////////
PARTNERLINKS
      List of services participating in this BPEL process
////////////////////////////////////////////////////////////////////
-->
<partnerLinks>
 <!--
  The 'client' role represents the requester of this service. It is
      used for callback. The location and correlation information
associated
      with the client role are automatically set using WS-Addressing.
    -->
    <partnerLink name="client" partnerLinkType="client:CreateInvoice"
                 myRole="CreateInvoiceProvider"/>
    <partnerLink name="CREATE_SINGLE_INVOICE_1037895"
                 partnerRole="CREATE_SINGLE_INVOICE_1037895_ptt_Role"
                 partnerLinkType="ns1:
CREATE_SINGLE_INVOICE_1037895_ptt_PL"/>
```

```
<parnterLink name="ReadPayload" partnerRole="SynchRead_role"
                partnerLinkType="ns4:SynchRead_plt"/>
</partnerLinks>
<!--
////////////////////////////////////////////////////////////////////
VARIABLES
      List of messages and XML documents used within this BPEL process
////////////////////////////////////////////////////////////////////
-->
<variables>
<!--Reference to the message passed as input during initiation-->
  <variable name="inputVariable"
              messageType="client:CreateInvoiceRequestMessage"/>
<!--Reference to the message that will be returned to the requester-->
    <variable name="outputVariable"
              messageType="client:CreateInvoiceResponseMessage"/>
    <variable name="
Invoke_1_CREATE_SINGLE_INVOICE_1037895_InputVariable"
              messageType="ns1:Request"/>
    <variable name="
Invoke_1_CREATE_SINGLE_INVOICE_1037895_OutputVariable"
              messageType="ns1:Response"/>
    <variable name="Invoke_2_SynchRead_InputVariable"
              messageType="ns4:Empty_msg"/>
    <variable name="Invoke_2_SynchRead_OutputVariable"
              messageType="ns4:InputParameters_msg"/>
</variables>
<!--
////////////////////////////////////////////////////////////////////
ORCHESTRATION LOGIC
     Set of activities coordinating the flow of messages across the
     services integrated within this business process
////////////////////////////////////////////////////////////////////
-->
<sequence name="main">
    <!--Receive input from requestor. (Note: This maps to operation
defined in CreateInvoice.wsdl)-->
  <receive name="receiveInput" partnerLink="client"
             portType="client:CreateInvoice" operation="process"
             variable="inputVariable" createInstance="yes"/>
    <!--Generate reply to synchronous request-->
    <assign name="SetHeader">
      <copy>
        <from expression="''operations'">
        <to variable="
Invoke_1_CREATE_SINGLE_INVOICE_1037895_InputVariable"
            part="header"
            query="/ns1:SOAHeader/ns2:ProcedureHeaderType/ns2:Username"
/>
      </copy>
      <copy>
        <from expression="''Receivables, Vision Operations (USA)'">
        <to variable="
Invoke_1_CREATE_SINGLE_INVOICE_1037895_InputVariable"
            part="header"
            query="/ns1:SOAHeader/ns2:ProcedureHeaderType/ns2:
Responsibility"/>
      </copy>
      <copy>
        <from expression="''204'">
        <to variable="
Invoke_1_CREATE_SINGLE_INVOICE_1037895_InputVariable"
            part="header"
            query="/ns1:SOAHeader/ns2:ProcedureHeaderType/ns2:ORG_ID"/>
      </copy>
      <copy>
```

```
<from expression="''Receivables, Vision Operations (USA)'">
        <to variable="
Invoke_1_CREATE_SINGLE_INVOICE_1037895_InputVariable"
            part="header"
            query="/ns1:SOAHeader/ns1:SecurityHeader/ns1:
ResponsibilityName"/>
      </copy>
   </assign>
   <invoke name="InvokeReadPayload" partnerLink="ReadPayload"
            portType="ns4:SynchRead_ptt" operation="SynchRead"
            inputVariable="Invoke_2_SynchRead_InputVariable"
            outputVariable="Invoke_2_SynchRead_OutputVariable"/>
   <assign name="SetPayload">
      <copy>
        <from variable="Invoke_2_SynchRead_OutputVariable"
              part="InputParameters" query="/ns3:InputParameters"/>
        Invoke_1_CREATE_SINGLE_INVOICE_1037895_InputVariable"
        part="body" query="/ns1:SOARequest/ns3:InputParameters"/>
      </copy>
   </assign>
   <assign name="SetDate">
      <copy>
        <from expression="xp20:current-date()">
        <to to variable="
Invoke_1_CREATE_SINGLE_INVOICE_1037895_InputVariable"
            part="body"
            query="/ns1:SOARequest/ns3:InputParameters/ns3:
P_TRX_HEADER_TBL/ns3:P_TRX_HEADER_TBL_ITEM/ns3:TRX_DATE"/>
      </copy>
   </assign>
   <invoke name="Invoke_1" partnerLink="CREATE_SINGLE_INVOICE_1037895"
            portType="ns1:CREATE_SINGLE_INVOICE_1037895_ptt"
            operation="CREATE_SINGLE_INVOICE_1037895"
            inputVariable="
Invoke_1_CREATE_SINGLE_INVOICE_1037895_InputVariable"
            outputVariable="
Invoke_1_CREATE_SINGLE_INVOICE_1037895_OutputVariable"/>
   <assign name="AssignResult">
      <copy>
        <from variable="
Invoke_1_CREATE_SINGLE_INVOICE_1037895_OutputVariable"
              part="body"
              query="/ns1:SOAResponse/ns3:OutputParameters/ns3:
X_MSG_DATA"/>
        <to variable="outputVariable" part="payload"
            query="/client:CreateInvoiceProcessResponse/client:result"/>
      </copy>
   </assign>
   <reply name="replyOutput" partnerLink="client"
           portType="client:CreateInvoice" operation="process"
           variable="outputVariable"/>
 </sequence>
</process>
```

Glossary of Annotations

This section includes a list of currently supported annotation types and details about their recommended use.

The following table describes the annotation information for `<description sentence(s)>`:

<description sentence(s)>

Annotation Type	<description sentence(s)>
Syntax	Does not require a tag.

Annotation Type	<description sentence(s)>
Usage	Defines a user-friendly description of what the interface or method does. Start the description with a summary sentence that begins with a capital letter and ends with a period. Do not use all capitals) and do not capitalize words that are not proper nouns. An example of a good beginning sentence could be as follows: "The Purchase Order Data Object holds the purchase order data including nested data objects such as lines and shipments." In general, a good description has multiple sentences and would be easily understood by a potential customer. An exception to the multiple sentence rule is cases where the package-level description provides detailed context information and the associated method-level descriptions can therefore be more brief (to avoid repetitiveness). A bad example would be: "Create an order." This description is barely usable. A better one would be: "Use this package to create a customer order, specifying header and line information." You can use the tag for forcing a new line in description. The following is an example on how to force a new line in the description: The following is an example on how to force a new line in the description: `FEM_BUDGETS_ATTR_T is an interface table for loading and updating Budget attribute assignments using the Dimension Member Loader. These attribute assignments are properties that further describe each Budget. When loading Budgets using the Dimension Member Loader, identify each new member in the FEM_BUDGETS_B_T table while providing an assignment row for each required attribute in the FEM_BUDGETS_ATTR_T table.`

Annotation Type	<description sentence(s)>

Example	`/*#`
	` * This is a sample description. Use`
	` * standard English capitalization and`
	` * punctuation. Write descriptions`
	` * carefully.`

Required	Required for all interfaces that have `@rep:scope public`.
Default	If not set, the value is defaulted from the Javadoc or PL/SQL Doc of the interface or method.
Level	Interface (class) and API (method).
Multiple Allowed	No. Use only one per each program element (class or method).

Comments	Optionally, you can use the following HTML tags in your descriptions:
	`<body>`
	`<p>`
	`<strong>`
	`<em>`
	`<ul>`
	`<li>`
	`<h1>`
	`<h2>`
	`<h3>`
	
	`<pre>` for multiple code samples (should be enclosed by `<code>` tags)

The following table describes the annotation information for `@rep:scope`:

@rep:scope

Annotation Type	@rep:scope
Syntax	`@rep:scope public | private | internal`
Usage	Indicates where to publish the interface, if at all.

Annotation Type	@rep:scope
Example	`@rep:scope public` means publish everywhere. **Note:** Public interfaces are displayed on the customer-facing UI. `@rep:scope private` means that this interface is published to the Integration Repository but restricted for use by the owning team. `@rep:scope internal` means publish within the company.
Required	Required for all interfaces.
Default	None.
Level	Interface (class) and API (method).
Multiple Allowed	No. Use only one per each program element (class or method).

The following table describes the annotation information for `@rep:product`:

@rep:product

Annotation Type	@rep:product
Syntax	`@rep:product` *`StringShortCode`*
Usage	Specifies the product shortname of the interface.
Example	`@rep:product PO`
Required	Required for all interfaces.
Default	None.

Annotation Type	@rep:product
Level	Interface (class) only.
Multiple Allowed	No. Use only one per interface.

The following table describes the annotation information for `@rep:implementation`:

@rep:implementation

Annotation Type	@rep:implementation
Syntax	`@rep:implementation` *`StringClassName`*
Usage	Specifies the implementation class name of the interface.
Example	`@rep:implementation oracle.apps.po.server.PurchaseOrdersAmImpl`
Required	Required for Java only.
Default	None.
Level	Interface (class).
Multiple Allowed	No. Use only one per interface.

The following table describes the annotation information for `@rep:displayname`:

@rep:displayname

Annotation Type	@rep:displayname
Syntax	`@rep:displayname` *`StringName`*
Usage	Defines a user-friendly name for the interface.
Example	`@rep:displayname Purchase Order Summary`

Annotation Type	@rep:displayname
Required	Required for all interfaces that have `@rep: scope public`.
Default	None.
Level	Interface (class) and API (method).
Multiple Allowed	No. Use only one per each program element (class or method).

Annotation Type	@rep:displayname
Comments	**Display Name Guidelines** These guidelines apply to display names for all technologies (interfaces, classes, methods, parameters, XMLG maps, and so on). Display names must meet the following criteria: • Be mixed case. Do not use all capitals or all lower case. • Be singular rather than plural. For example, use "Customer" instead of "Customers". • Be fully qualified and representative of your business area. • Not have underscores (_). • Not end with a period (.). • Not be the same as the internal name. • Not begin with a product code or product name. • Not contain obvious redundancies such as "Package", "API", or "APIs". As you write your display names, do consider the UI where the display name will be seen. For example, use 'Promise Activity' as the display name, instead of IEX_PROMISES_PUB. The reason is that IEX_PROMISES_PUB contains underscores and is the same as the internal name. Use 'Process Activity' as the display name, instead of 'Workflow Process Activity APIs'. This is because it begins with a product name and ends with "APIs".

The following table describes the annotation information for `@rep:lifecycle`:

@rep:lifecycle

Annotation Type	@rep:lifecycle
Syntax	`@rep:lifecycle active | deprecated | obsolete | planned`
Usage	Indicates the lifecycle phase of the interface.
Example	`@rep:lifecycle active` means the interface is active. `@rep:lifecycle deprecated` means the interface has been deprecated. `@rep:lifecycle obsolete` means the interface is obsolete and must not be used. `@rep:lifecycle planned` means the interface is planned for a future release. This is used for prototypes and mockups.
Required	Optional.
Default	The default value is `active`.
Level	Interface (class) and API (method).
Multiple Allowed	No. Use only one per each program element (class or method).
Comments	The parsers will validate that this annotation is in sync with the "@deprecated" Javadoc annotation.

The following table describes the annotation information for `@rep:compatibility`:

@rep:compatibility

Annotation Type	@rep:compatibility
Syntax	`@rep:compatibility S | N`

Annotation Type	@rep:compatibility
Usage	`S` indicates the lifecycle phase of the interface. `N` indicates that backward compatibility is not assured.
Example	`@rep:compatibility S`
Required	Optional.
Default	Conditional. The value is defaulted to `S` for `@rep:scope public`. Otherwise, the value is defaulted to `N`.
Level	Interface (class) and API (method).
Multiple Allowed	No. Use only one per each program element (class or method).

The following table describes the annotation information for `@link`:

@link

Annotation Type	@link **Note: This is supported only for a destination of Java.**
Syntax	`{@link package.class#member label}`
Usage	Provides a link to another interface or method.
Example	`{@link #setAmounts(int,int,int, int) Set Amounts}`
Required	Optional.
Default	None.
Level	Interface (class) and API (method).

Annotation Type	@link **Note: This is supported only for a destination of Java.**
Multiple Allowed	Yes.
Comments	This is the standard Javadoc "@link" annotation, where the linked items are embedded as hyperlinks in the description that displays in the UI. Take note of the following rules: Public APIs must not link to private or internal APIs. @link annotations must not link to documents that are not accessible by the Integration Repository viewer.

The following table describes the annotation information for `@see`:

@see

Annotation Type	@see
Syntax	`@see StringLocator`
Usage	Provides a link to another interface or method.
Example	`@see #setAmounts(int,int,int,int)`
Required	Optional.
Default	None.
Level	Interface (class) and API (method).
Multiple Allowed	Yes.

Annotation Type	@see
Comments	This is the standard Javadoc "@see" annotation. The linked items will display on the UI under a "See Also" heading. **Usage in PL/SQL Code: @see package#procedure**

The following table describes the annotation information for `@rep:ihelp`:

@rep:ihelp

Annotation Type	@rep:ihelp

Syntax	When used as a separate child annotation on a single line:
	
	`@rep:ihelp <product_shortname>/@<help_target> #<help_target> <link_text>`
	
	When used as an inline annotation, add curly braces:
	
	`{@rep:ihelp <product_shortname>/@<help_target> #<help_target> <link_text>}`

Usage	Provides a link to an existing HTML online help page. `product_shortname` is the product short name. `help_target` is the help target that was manually embedded in the file by the technical writer, such as, "jtfaccsum_jsp," "aolpo," "overview," "ast_aboutcollateral". For more information on how to customize Oracle E-Business Suite help, see Setting Up Oracle E-Business Suite Help, *Oracle E-Business Suite Setup Guide*.
Example	`@rep:ihelp #setAmounts(int,int, int,int)`

Annotation Type	@rep:ihelp
Required	Optional.
Default	None.
Level	Interface (class) and API (method).
Multiple Allowed	Yes.

The following table describes the annotation information for `@rep:metalink`:

@rep:metalink

Annotation Type	@rep:metalink

Syntax	When used as a separate child annotation on a single line:
	`@rep:metalink <bulletin_number> <link_text>`
	When used as an inline annotation, add curly braces:
	`{@rep:metalink <bulletin_number> <link_text>}`

Usage	Provides a link to an existing My Oracle Support (formerly Oracle*MetaLink*) Knowledge Document.
Example	`@rep:metalink 123456.1 See My Oracle Support Knowledge Document 123456.1`
Required	Optional.
Default	None.
Level	Interface (class) and API (method).
Multiple Allowed	Yes.

The following table describes the annotation information for `@rep:category`:

@rep:category

Annotation Type	@rep:category
Syntax	• `@rep:category BUSINESS_ENTITY BUSINESS_ENTITY_CODE` • `@rep:category IREP_CLASS_SUBTYPE JAVA_BEAN_SERVICES` • `@rep:category IREP_CLASS_SUBTYPE AM_SERVICES`
Usage	• Specifies the business category of the interface. • Indicates a Java API as a serviceable interface. • Indicates an Application Module class as a serviceable interface.
Example	• `@rep:category BUSINESS_ENTITY PO_PLANNED_PURCHASE_ORDER` PO_PLANNED_PURCHASE_ORDER is your business entity code and your display name for example could be "Planned Purchase Order". • `@rep:category IREP_CLASS_SUBTYPE JAVA_BEAN_SERVICES` "Java Bean Services" can be displayed as a subtype of a Java API. • `@rep:category IREP_CLASS_SUBTYPE AM_SERVICES` "Application Module Services" can be displayed as a subtype of a Java API. See Business Entity Annotation Guidelines, page A-41 for additional details.

Annotation Type	@rep:category
Required	• `BUSINESS_ENTITY` is mandatory for all interfaces. If the methods belonging to a class ALL have the same business entity, you only need to annotate the class. However, if the methods belonging to a class have heterogeneous business entities, then you have to annotate each of the methods appropriately. • `IREP_CLASS_SUBTYPE JAVA_BEAN_SERVICES` annotation is mandatory for a Java interface that needs to be identified as a serviceable Java API. • `IREP_CLASS_SUBTYPE AM_SERVICES` annotation is mandatory for an Application Module that needs to be identified as a serviceable API.
Default	None
Level	`BUSINESS_ENTITY` is applicable for both class level and method level. However, `JAVA_BEAN_SERVICES` and `AM_SERVICES` are applicable only for class level.
Multiple Allowed	Yes.
Comments	You are encouraged to use the `rep:category` annotation liberally in your code.

The following table describes the annotation information for `@rep:usestable`:

@rep:usestable

Annotation Type	@rep:usestable
Syntax	`@rep:usestable <table or view name> <sequence> <direction flag>`

Annotation Type	@rep:usestable
Usage	Used when annotating concurrent programs to identify associated open interface tables or open interface views. `<table or view name>` is the name of the table or view. `<sequence>` is an integer used to tell the UI the display order of the different pieces. By convention, in the rep: category OPEN_INTERFACE, page A-128 annotation, you will have used `1` for the concurrent program. Here in the `rep:usestable` annotations, order the input tables: list primary (header) tables before detail (lines) tables. Finally, put any output views or tables at the end of the sequence. `<direction flag>` is optional and specifies one of the following: `IN` (default), `OUT`, or `BOTH`.
Example	`@rep:usestable SampleTable 3 IN`
Required	Only if the concurrent program is part of an open interface.
Default	None.
Level	Interface.
Multiple Allowed	Yes.

The following table describes the annotation information for `@rep:standard`:

@rep:standard

Annotation Type	@rep:standard
Syntax	`@rep:standard` *`StringType StringVersionNumber StringSpecName`* In the following example `@rep:standard OAG 7.2 Process_PO_001` StringType is OAG, StringVersionNumber is 7.2 and StringSpecName is Process_PO_001 See Annotation Syntax, page A-1 for details about this annotation's syntax.

Annotation Type	@rep:standard
Usage	Specifies the business standard name. This annotation is reserved for where Oracle is compliant with industry standards.
Example	In the example `@rep:standard RosettaNet 02.02.00 'Pip3B12-Shipping Order Confirmation`, the StringSpecName is enclosed in Single Quotes because the spec name has empty spaces. It is not necessary to have these quotes if the StringSpecName does not have any empty spaces like the following example `@rep:standard RosettaNet 02.02.00 Pip3B12-PurchaseOrderConfirmation`.
Required	Optional.
Default	Methods default to the value set on the class.
Level	Documents and data rows.
Multiple Allowed	No. Use only one per each program element (class or method).

The following table describes the annotation information for `@rep:httpverb`:

@rep:httpverb

Annotation Type	@rep:httpverb
Syntax	`@rep:httpverb <HTTP_Method_Types>` Use a comma separated list of the HTTP verbs (GET and POST) at the method level. **Note:** GET and POST are the supported HTTP methods for Java Bean Services and Application Module Services in this release.

Annotation Type	@rep:httpverb
Usage	Use this annotation to indicate the HTTP Verbs suitable for the current method or operation. If a method is not annotated with POST `httpverb`, the POST check box is still active by default in the selected interface details page. This allows an integration administrator to select the POST verb for that method if needed before service deployment. However, unlike the POST HTTP verb, if a method is not annotated with GET `httpverb`, then the GET check box becomes inactive or disabled for that method. This means that method will not be exposed as REST service with GET operation. This annotation is available for Java files only.

Example	The comma separate list can be used in the following ways:
	
	`@rep:httpverb get`
	
	`@rep:httpverb post`
	
	`@rep:httpverb get, post`

Required	Optional.
Default	None.
Level	Method level
Multiple Allowed	No. Use only one per method.
Comments	Use this annotation to optimize the HTTP method such as GET and POST.

The following table describes the annotation information for `@rep:interface`:

@rep:interface

Annotation Type	@rep:interface
Syntax	`@rep:interface StringClassName where the StringClassName syntax is transactiontype: subtype. Refer to the example below.`

Annotation Type	@rep:interface
Usage	Specifies the interface name for technologies where parsing tools can't easily introspect the interface name.
Example	The `StringClassName` is always `transactiontype:subtype` `@rep:interface PO:POC`
Required	Optional.
Default	None.
Level	Interface only.
Multiple Allowed	No. Use only one per interface.
Comments	Used in technologies where there isn't a strong native definition of the interface, such as XML Gateway and e-Commerce Gateway

The following table describes the annotation information for `@param`:

@param

Annotation Type	@param
Syntax	`@param paramName paramDescription` Ensure that all parameters have descriptions and the parameter names must not contain spaces.
Usage	Specifies the name and description of a method, procedure, or function parameter (IN, OUT, or both).
Example	`@param PONumber The purchase order number.`
Required	Optional.
Default	None.
Level	Methods, procedures and functions.

Annotation Type	@param
Multiple Allowed	Yes.
Comments	For convenience, Java annotations are also supported.

The following table describes the annotation information for `@return`:

@return

Annotation Type	@return
Syntax	`@return StringDescription`
Usage	Specifies the description of a method or function return parameter.
Example	`@return The purchase order status.`
Required	Optional.
Default	None.
Level	Methods, procedures and functions.
Multiple Allowed	Yes.
Comments	For convenience, Java annotations are also supported.

The following table describes the annotation information for `@rep:paraminfo`:

@rep:paraminfo

Annotation Type	@rep:paraminfo
Syntax	`@rep:paraminfo {@rep:innertype typeName} {@rep:precision value} {@rep:required} {@rep:key_param}`

Annotation Type	@rep:paraminfo

Usage	`rep:paraminfo`
	
	The rep:paraminfo annotation must come immediately in the line following the parameter's `@param` or `@return` annotation it is describing.
	
	`rep:innertype`
	
	Optional inline annotation to describe the inner type of generic objects such as collections.
	
	`rep:precision`
	
	Optional inline annotation to specify the parameter precision. Used for Strings and numbers.
	
	`rep:required`
	
	Optional inline annotation to indicate that a not null must be supplied. This is only needed for non-PL/SQL technologies.
	
	`rep:key_param`
	
	Optional inline annotation to define a parameter as a key parameter or path variable for REST services. It is applicable for Java or Application Module methods. If this annotation is used, then `rep:required` must be present. Additionally, this annotation is not applicable to `rep:paraminfo` after `@return` annotation.

Example	`/**`
	` * Gets the price for a purchase order line item.`
	` *`
	` * @param poNumber purchase order unique identifier`
	` * @paraminfo {@rep:precision 10} {@rep:required} {@rep:key_param}`
	` * @param lineNumber purchase order line unique identifier`
	` * @paraminfo {@rep:precision 10} {@rep:required}`
	` * @return the item price for the given purchase order line`
	` * @paraminfo {@rep:precision 10}`
	` *`
	` * @rep:scope public`
	` * @rep:displayname Get Purchase Order Line Item Price`
	` */`
	`public Number getItemPrice(Number poNumber, Number lineNumber);`

Required	Optional.

Annotation Type	@rep:paraminfo
Default	None.
Level	Methods only.
Multiple Allowed	Yes. Multiple values can be assigned for different parameters.

The following table describes the annotation information for `@rep:businessevent`:

@rep:businessevent

Annotation Type	@rep:businessevent
Syntax	`@rep:businessevent BusinessEvent`
Usage	Indicates the name of the business event raised by this method.
Example	`@rep:businessevent oracle.apps.wf. notification.send`
Required	Optional.
Default	Defaulted in file types where the business event can be derived.
Level	Methods only.
Multiple Allowed	Yes.
Comments	Make sure to use this annotation at every instance where you raise a business event. Note that business events themselves do not require an annotation.

The following table describes the annotation information for `@rep:direction`:

@rep:direction

Annotation Type	@rep:direction
Syntax	`@rep:direction <OUT | IN>`
Usage	Indicates whether the interface is outbound or inbound.
Example	`@rep:direction OUT`
Required	Required for EDI and XML Gateway annotations only.
Default	None.
Level	Interface.
Multiple Allowed	No.

The following table describes the annotation information for `@rep:service`:

@rep:service

Annotation Type	@rep:service
Syntax	`@rep:service`
Usage	Indicates that a Java file is a business service object (BSO). Use this tag as it is in your Java file. Refer to the Example section below. It takes no parameters.

Annotation Type	@rep:service
Example	/** * The Purchase Order service lets you to view, update, acknowledge and * approve purchase orders. It also lets you receive items, and obtain * pricing by line item. * * @see oracle.apps.fnd.framework.toolbox.tutorial.PurchaseOrderSDO * @see oracle.apps.fnd.framework.toolbox.tutorial.PurchaseOrderAcknowledgementsSDO * @see oracle.apps.fnd.framework.toolbox.tutorial.PurchaseOrderReceiptsSDO * * @rep:scope public * @rep:displayname Purchase Order Service * @rep:implementation oracle.apps.fnd.framework.toolbox.tutorial.server.PurchaseOrderSAMImpl * @rep:product PO * @rep:category BUSINESS_ENTITY PO_PURCHASE_ORDER *** @rep:service** */
Required	Required for Business Service Objects.
Default	None.
Level	Class.
Multiple Allowed	No.

The following table describes the annotation information for `@rep:servicedoc`:

@rep:servicedoc

Annotation Type	@rep:servicedoc
Syntax	`@rep:servicedoc`
Usage	Indicates that a Java file is an SDO (as opposed to a normal Java API). Use this tag as is in your java file. Refer to the example section below. It takes no parameters.

Annotation Type	@rep:servicedoc
Example	/** * The Purchase Order Data Object holds the purchase order data including * nested data objects such as lines and shipments. * * @see oracle.apps.fnd.framework.toolbox.tutorial.PurchaseOrderLineSDO * * @rep:scope public * @rep:displayname Purchase Order Data Object * @rep:product PO * @rep:category BUSINESS_ENTITY PO_PURCHASE_ORDER *** @rep:servicedoc** */
Required	Required for Service Data Objects.
Default	None.
Level	Class.
Multiple Allowed	No.
Comments	Developers do not need to enter this annotation because it is automatically generated.

The following table describes the annotation information for `@rep:synchronicity`:

@rep:synchronicity

Annotation Type	@rep:synchronicity
Syntax	`@rep:synchronicity <SYNCH or ASYNCH>`
Usage	Specifies synchronous or asynchronous behavior.
Example	`@rep:synchronicity SYNCH`
Required	Optional.
Default	Is defaulted based on module type. For example, `ASYNCH` for XML Gateway and `SYNCH` for Business Service Object.

Annotation Type	@rep:synchronicity
Level	Class or method.
Multiple Allowed	No.

The following table describes the annotation information for `@rep:appscontext`:

@rep:appscontext

Annotation Type	@rep:appscontext
Syntax	`@rep:appscontext <NONE, APPL, RESP, USER, NLS, or ORG>`
Usage	Specifies the context required to execute the method.
Example	`@rep:appscontext USER`
Required	Optional.
Default	`NONE`
Level	Method.
Multiple Allowed	No, only one allowed per method.

The following table describes the annotation information for `@rep:comment`:

@rep:comment

Annotation Type	@rep:comment
Syntax	`@rep:comment <comment>`
Usage	This annotation is skipped by the parsers. It is for use by product teams when a non-published comment is desired.
Example	`@rep:comment This is a sample comment.`

Annotation Type	@rep:comment
Required	Optional.
Default	None.
Level	Any.

The following table describes the annotation information for `@rep:primaryinstance` :

@rep:primaryinstance

Annotation Type	@rep:primaryinstance
Syntax	`@rep:primaryinstance`
Usage	To indicate the primary instance of an overloaded method or procedure.
Required	Required for all overloaded methods and procedures.
Default	None.
Level	Method or procedure.
Multiple Allowed	No.
Comments	The primary instance's display name and description will be used in the browser UI when a list of methods is displayed. The non-primary instances (such as, the overloads) should have descriptions that emphasize how they differ from the primary (such as, "This variant allows specification of the org_id."). The non-primary display names and descriptions will only be displayed when viewing the details of the overloaded interface.

The following table describes the annotation information for `@rep:usesmap`:

@rep:usesmap

Annotation Type	**@rep:usesmap**
Syntax	`@rep:usesmap <map_name> <sequence_number>`
Usage	To indicate the E-Commerce Gateway maps that are associated with a concurrent program. `<map_name>` where map_name is the default map name. `<sequence_number>` is an integer used to tell the UI the display order of the different pieces.
Example	`@rep:usesemap SampleMap 2`
Required	Optional.
Default	None.
Level	Any.
Multiple Allowed	Yes.
Comments	The default map name has the following naming convention "EC_XXXX_FF" where XXXX is the 4-letter acronym for your transaction.

B

Configuring Server Connection

Overview

If your web services are exposed and invoked through BPEL PM, to successfully deploy BPEL processes to Oracle Application Server, you must first establish the necessary server connection information used at runtime in the background.

This chapter includes the following topics on how to configure the server connection:

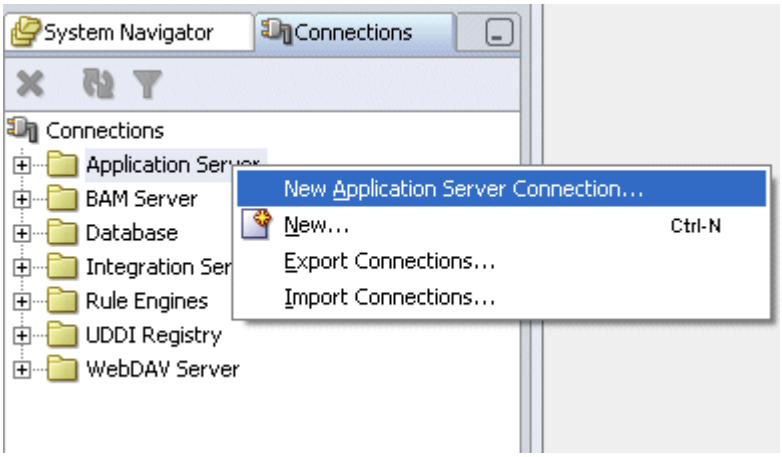
Creating the Application Server Connection

You must establish a connectivity between the design-time environment and the server you want to deploy it to. In order to establish such a connectivity, you must create the application server connection.

Use the following steps to create the application server connection:

1. From Oracle JDeveloper (for example, Oracle JDeveloper 10.1.3.3), select **View** > **Connection Navigator** to open the Connections tab.
2. Right-click on the Application Server and select **New Application Server Connection**.

Oracle JDeveloper: Connections Tab

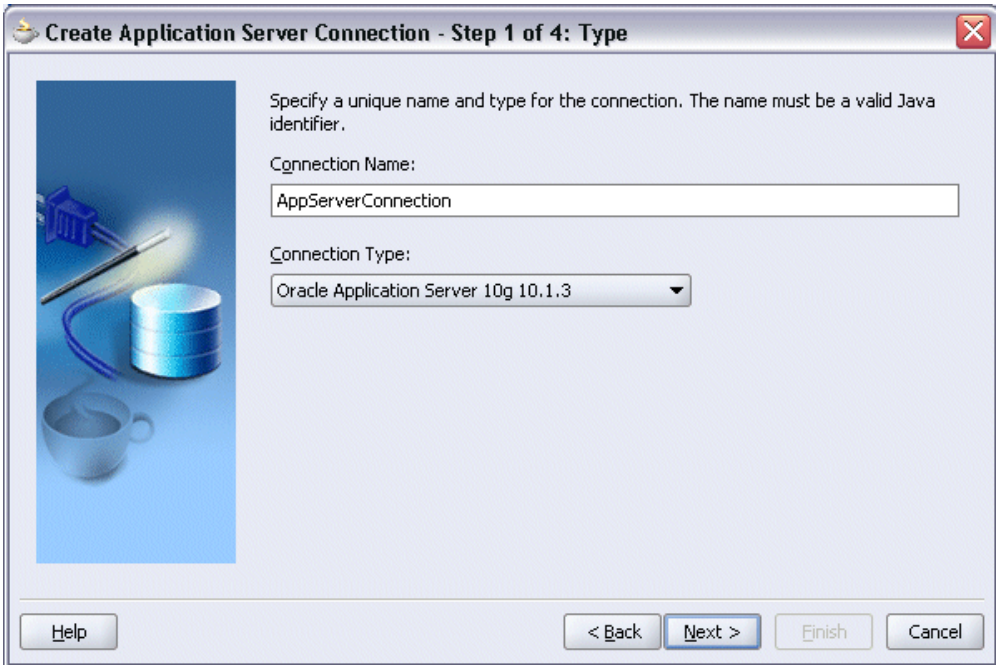


This opens the Create Application Server Connection wizard. Click **Next** in the Welcome page of the wizard.

3. Enter the connection name and select **Oracle Application Server 10g 10.1.3** as the connection type.

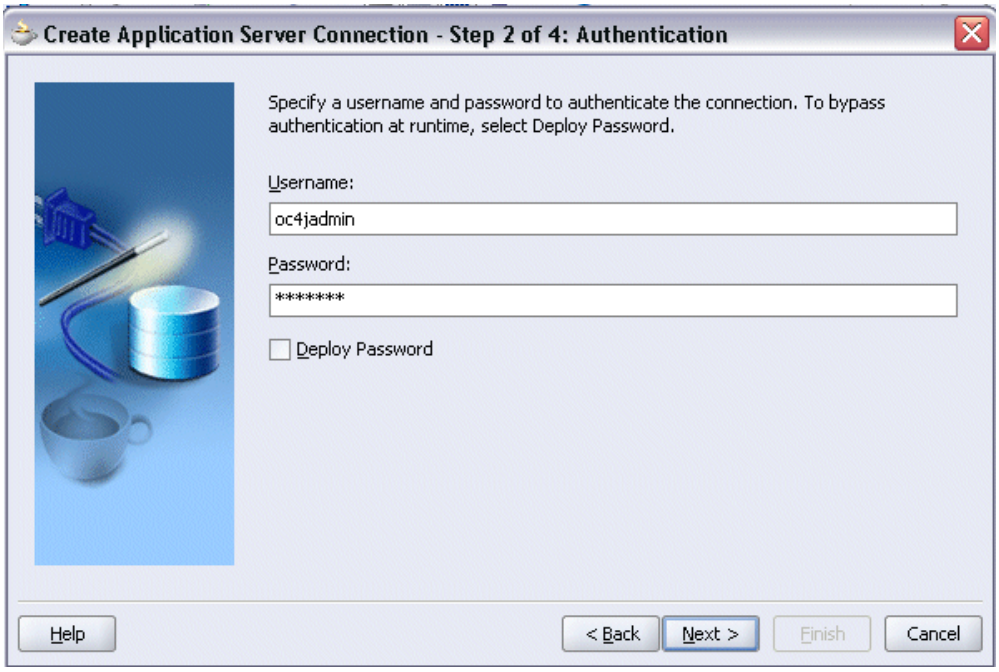
   Click **Next**.

Create Application Server Connection - Type Dialog



4. Enter a valid username (such as `oc4jadmin`) and the password information and click **Next**.

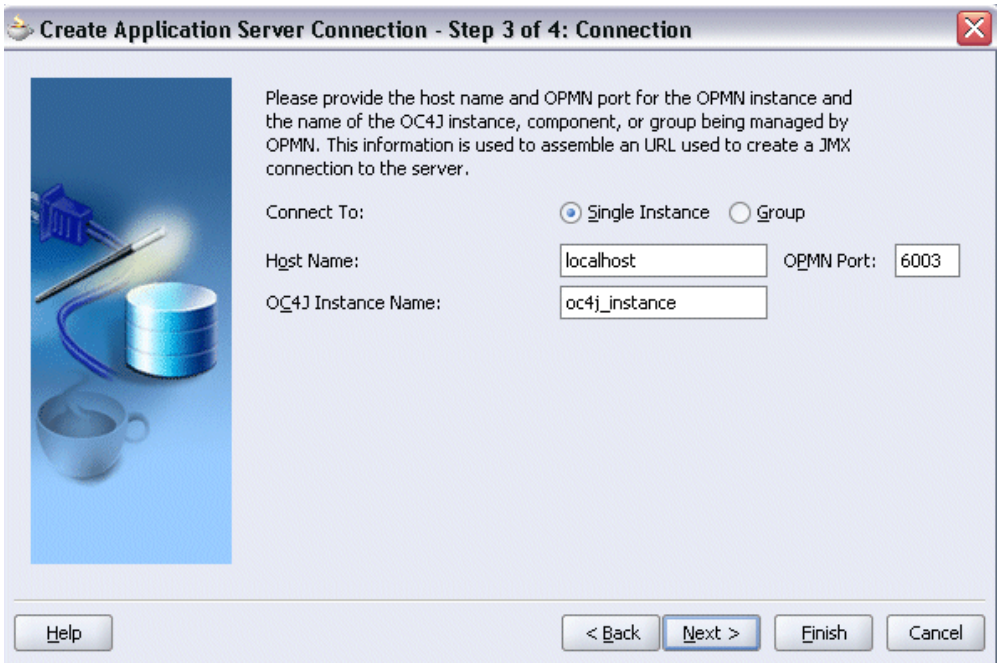
Create Application Server Connection - Authentication Dialog



5. Select **Single Instance** radio button. Enter appropriate values for the Server connection host name, port, and OC4J instance information.

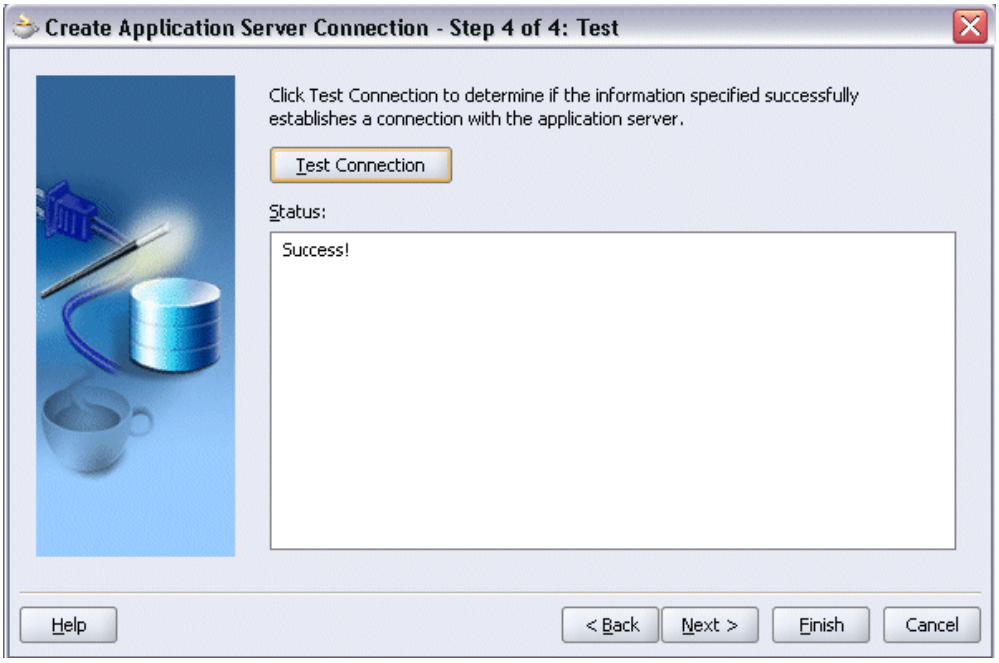
   Click **Next**.

Create Application Server Connection - Connection Dialog



6. Click **Test Connection** to validate your server configuration. You should find "Success!" populated in the Status window.

Create Application Server Connection - Test Dialog



Click **Finish**.

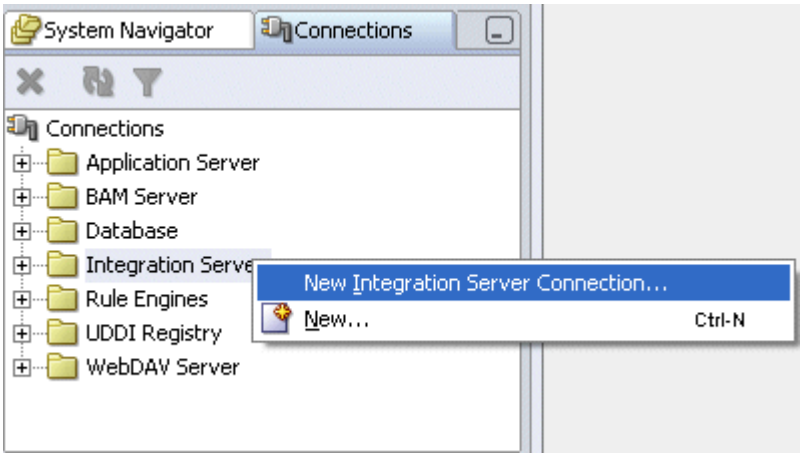
Creating the Integration Server Connection

Use the following steps to create the integration server connection:

1. From Oracle JDeveloper, select **View >Connection Navigator** to open the Connections tab.

2. Right-click on the Integration Server and select **New Integration Server Connection** .

Oracle JDeveloper: Connections Tab

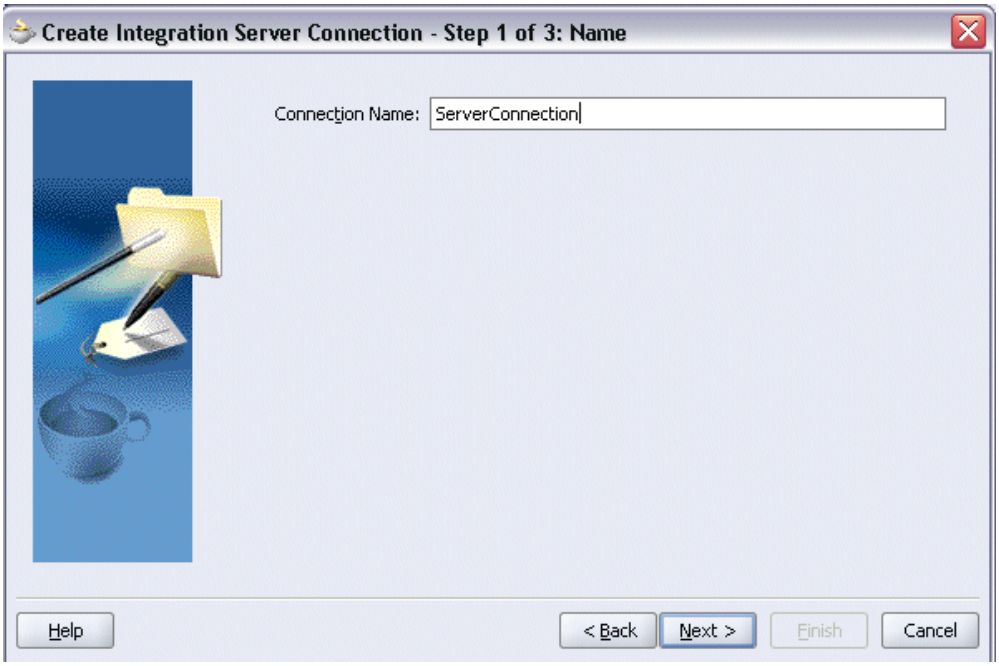


This opens the Create Integration Server Connection wizard. Click **Next** in the Welcome page of the wizard.

3. Enter the connection name.

   Click **Next**.

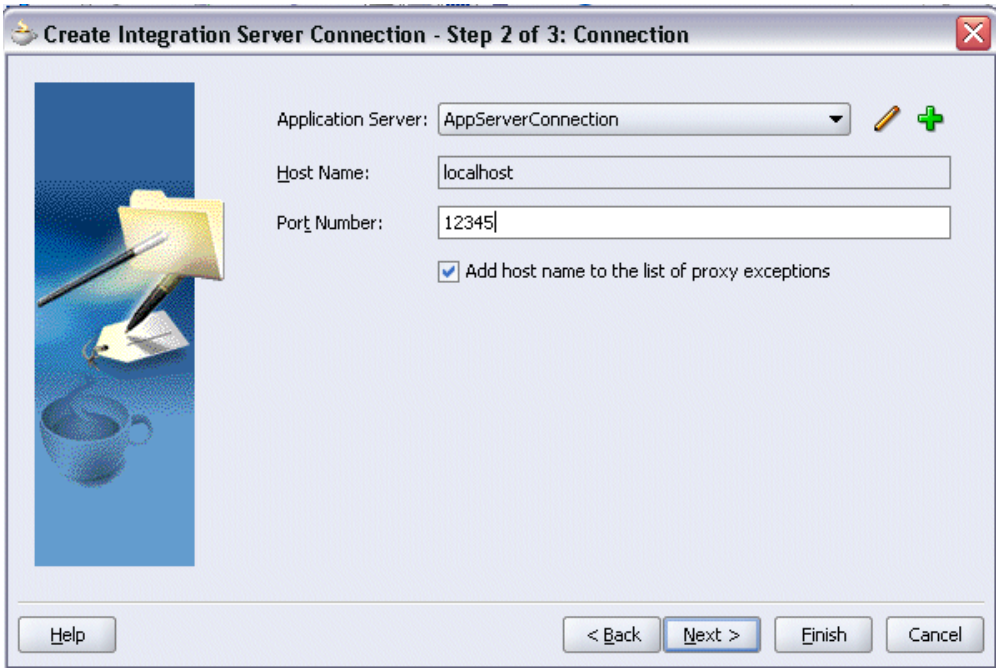
Create Integration Server Connection - Name Dialog



4. Select the Application Server name you just created from the drop-down list. The Host Name field will be populated automatically based on your selection. Enter HTTP port number.

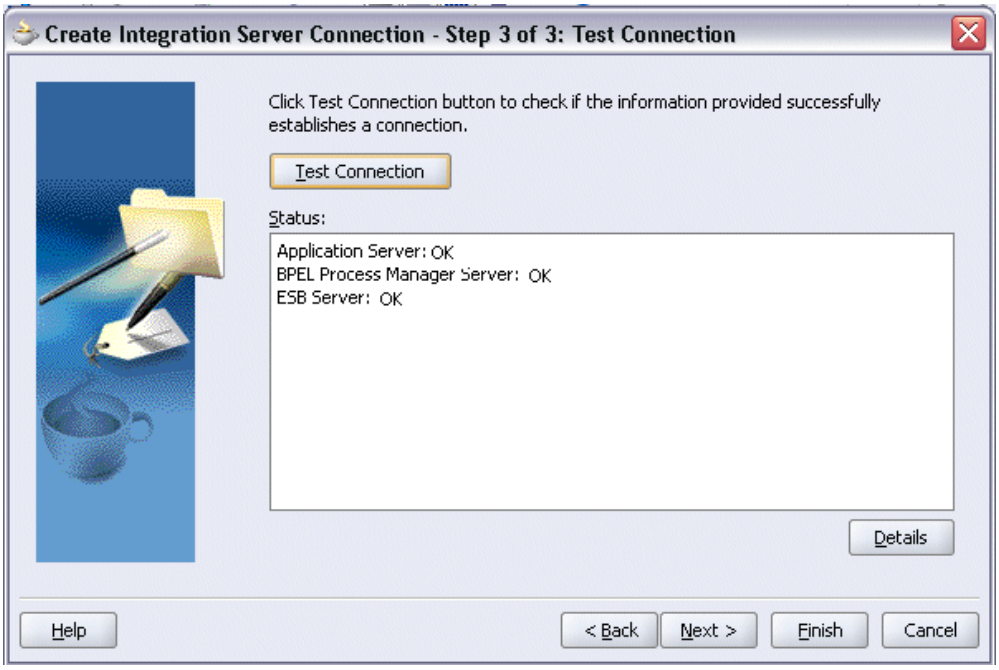
   Click **Next**.

Create Integration Server Connection - Connection Dialog



5. Click **Test Connection** to validate your integration server connection. You should find success messages populated in the Status window.

Create Integration Server Connection - Test Connection Dialog



Click **Finish**.

C

Sample Payload

Sample Payload for Creating Supplier Ship and Debit Request

The following information shows the sample payload in the `InputCreateSDRequest.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
 <cre:InputParameters  xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:cre="http://xmlns.oracle.
com/apps/ozf/soaprovider/plsql/ozf_sd_request_pub/create_sd_request/">
     <cre:P_API_VERSION_NUMBER>1.0</cre:P_API_VERSION_NUMBER>
  <cre:P_INIT_MSG_LIST>T</cre:P_INIT_MSG_LIST>
     <cre:P_COMMIT>F</cre:P_COMMIT>
  <cre:P_VALIDATION_LEVEL>100</cre:P_VALIDATION_LEVEL>
       <cre:P_SDR_HDR_REC>
         <cre:REQUEST_NUMBER>SDR-CREATE-BPEL1</cre:REQUEST_NUMBER>
         <cre:REQUEST_START_DATE>2008-08-18T12:00:00</cre:
REQUEST_START_DATE>
       <cre:REQUEST_END_DATE>2008-10-18T12:00:00</cre:REQUEST_END_DATE>>
         <cre:USER_STATUS_ID>1701</cre:USER_STATUS_ID>
         <cre:REQUEST_OUTCOME>IN_PROGRESS</cre:REQUEST_OUTCOME>
         <cre:REQUEST_CURRENCY_CODE>USD</cre:REQUEST_CURRENCY_CODE>
         <cre:SUPPLIER_ID>601</cre:SUPPLIER_ID>
         <cre:SUPPLIER_SITE_ID>1415</cre:SUPPLIER_SITE_ID>
         <cre:REQUESTOR_ID>xxxxxxxxx</cre:REQUESTOR_ID>
      <cre:ASSIGNEE_RESOURCE_ID>xxxxxxxxx</cre:ASSIGNEE_RESOURCE_ID>
      <cre:ORG_ID>204</cre:ORG_ID>
      <cre:ACCRUAL_TYPE>SUPPLIER</cre:ACCRUAL_TYPE>
      <cre:REQUEST_DESCRIPTION>Create</cre:REQUEST_DESCRIPTION>
         <cre:SUPPLIER_CONTACT_EMAIL_ADDRESS>sdr.supplier@example.
com</cre:SUPPLIER_CONTACT_EMAIL_ADDRESS>
         <cre:SUPPLIER_CONTACT_PHONE_NUMBER>2255</cre:
SUPPLIER_CONTACT_PHONE_NUMBER>
       <cre:REQUEST_TYPE_SETUP_ID>400</cre:REQUEST_TYPE_SETUP_ID>
         <cre:REQUEST_BASIS>Y</cre:REQUEST_BASIS>
         <cre:USER_ID>xxxxxxx</cre:USER_ID>
   </cre:P_SDR_HDR_REC>
   <cre:P_SDR_LINES_TBL>
       <cre:P_SDR_LINES_TBL_ITEM>
          <cre:PRODUCT_CONTEXT>PRODUCT</cre:PRODUCT_CONTEXT>
      <cre:INVENTORY_ITEM_ID>2155</cre:INVENTORY_ITEM_ID>
      <cre:ITEM_UOM>Ea</cre:ITEM_UOM>
      <cre:REQUESTED_DISCOUNT_TYPE>%</cre:REQUESTED_DISCOUNT_TYPE>
      <cre:REQUESTED_DISCOUNT_VALUE>20</cre:REQUESTED_DISCOUNT_VALUE>
      <cre:COST_BASIS>200</cre:COST_BASIS>
      <cre:MAX_QTY>200</cre:MAX_QTY>
      <cre:APPROVED_DISCOUNT_TYPE>%</cre:APPROVED_DISCOUNT_TYPE>
      <cre:APPROVED_DISCOUNT_VALUE>20</cre:APPROVED_DISCOUNT_VALUE>
      <cre:APPROVED_MAX_QTY>200</cre:APPROVED_MAX_QTY>
      <cre:VENDOR_APPROVED_FLAG>Y</cre:VENDOR_APPROVED_FLAG>
      <cre:PRODUCT_COST_CURRENCY>USD</cre:PRODUCT_COST_CURRENCY>
      <cre:END_CUSTOMER_CURRENCY>USD</cre:END_CUSTOMER_CURRENCY>
         </cre:P_SDR_LINES_TBL_ITEM>
      </cre:P_SDR_LINES_TBL>
   <cre:P_SDR_CUST_TBL>
   <cre:P_SDR_CUST_TBL_ITEM>
      <cre:CUST_ACCOUNT_ID>1290</cre:CUST_ACCOUNT_ID>
      <cre:PARTY_ID>1290</cre:PARTY_ID>
      <cre:SITE_USE_ID>10479</cre:SITE_USE_ID>
      <cre:CUST_USAGE_CODE>BILL_TO</cre:CUST_USAGE_CODE>
      <cre:END_CUSTOMER_FLAG>N</cre:END_CUSTOMER_FLAG>
      </cre:P_SDR_CUST_TBL>
         <cre:P_SDR_CUST_TBL_ITEM>
      <cre:CUST_ACCOUNT_ID>1287</cre:CUST_ACCOUNT_ID>
      <cre:PARTY_ID>1287</cre:PARTY_ID>
      <cre:SITE_USE_ID>1418</cre:SITE_USE_ID>
      <cre:CUST_USAGE_CODE>CUSTOMER</cre:CUST_USAGE_CODE>
      <cre:END_CUSTOMER_FLAG>Y</cre:END_CUSTOMER_FLAG>
      </cre:P_SDR_CUST_TBL>
  </cre:P_SDR_CUST_TBL>
</cre:InputParameters>
```

Sample Payload for Inbound Process Purchase Order XML Transaction

The following information shows the sample payload in the `order_data_xmlg.xml` file:

```
<?xml version="1.0">
  <PROCESS_PO_007> <!--xmlns="http://example.com/ServiceName"-->
    <CNTROLAREA>
        <BSR>
     <VERB>PROCESS</VERB>
           <NOUN>PO</NOUN>
    <REVISION>007</REVISION>
   </BSR>
   <SENDER>
    <LOGICALID/>
    <COMPONENT>BPEL</COMPONENT>
    <TASK>POISSUE</TASK>
    <REFERENCEID>refid</REFERENCEID>
    <CONFIRMATION>2</CONFIRMATION>
    <LANGUAGE>ENG</LANGUAGE>
    <CODEPAGE>US7ASCII</CODEPAGE>
    <AUTHID>APPS</AUTHID>
   </SENDER>
   <DATETIME qualifier="CREATION">
    <YEAR>2002</YEAR>
    <MONTH>10</MONTH>
    <DAY>09</DAY>
    <HOUR>16</HOUR>
    <MINUTE>45</MINUTE>
    <SECOND>47</SECOND>
    <SUBSECOND>356</SUBSECOND>
    <TIMEZONE>-0800</TIMEZONE>
   </DATETIME>
  </CNTROLAREA>
  <DATAAREA>
   <PROCESS_PO>
    <POORDERHDR>
     <DATETIME qualifier="DOCUMENT">
      <YEAR>2002</YEAR>
      <MONTH>10</MONTH>
      <DAY>09</DAY>
      <HOUR>16</HOUR>
      <MINUTE>40</MINUTE>
      <SECOND>34</SECOND>
      <SUBSECOND>000</SUBSECOND>
      <TIMEZONE>+0100</TIMEZONE>
     </DATETIME>
     <OPERAMT qualifier="EXTENDED" type="T">
      <VALUE>107.86</VALUE>
      <NUMOFDEC>6</NUMOFDEC>
      <SIGN>+</SIGN>
      <CURRENCY>USD</CURRENCY>
      <UOMVALUE>1</UOMVALUE>
      <UOMNUMDEC>0</UOMNUMDEC>
      <UOM>Ea</UOM>
   </OPERAMT>
    <POID>refid</POID>
    <POTYPE>Mixed</POTYPE>
    <CONTRACTS/>
    <DESCRIPTN/>
    <NOTES index="1"/>
    <USERAREA/>
    <PARTNER>
      <NAME index="1"/>
      <ONETIME>0</ONETIME>
      <PARTNRID/>
      <PARTNRTYPE>SoldTo</PARTNRTYPE>
      <PARTNRIDX>Example-01</PARTNRIDX>
    </PARTNER>
 </POORDERHDR>
  <POORDERLIN>
```

```
<QUANTITY qualifier="ORDERED">
      <VALUE>1</VALUE>
      <NUMOFDEC>0</NUMOFDEC>
      <SIGN>+</SIGN>
      <UOM>Ea</UOM>
  </QUANTITY>
   <OPERAMT qualifier="UNIT" type="T">
      <VALUE>107.86</VALUE>
      <NUMOFDEC>6</NUMOFDEC>
      <SIGN>+</SIGN>
      <CURRENCY>USD</CURRENCY>
      <UOMVALUE>1</UOMVALUE>
      <UOMNUMDEC>0</UOMNUMDEC>
      <UOM>Ea</UOM>
  </OPERAMT>
   <POLINENUM>1</POLINENUM>
   <ITEMRV/>
   <NOTES index="1"/>
   <ITEM>LAP-DLX</ITEM>
   <POLINESCHD>
      <DATETIME qualifier="NEEDDELV">
        <YEAR>2002</YEAR>
        <MONTH>10</MONTH>
        <DAY>09</DAY>
        <HOUR>00</HOUR>
        <MINUTE>00</MINUTE>
        <SECOND>00</SECOND>
        <SUBSECOND>000</SUBSECOND>
        <TIMEZONE>+0100</TIMEZONE>
   </DATETIME>
      <QUANTITY qualifier="ORDERED">
        <VALUE>1</VALUE>
        <NUMOFDEC>0</NUMOFDEC>
        <SIGN>+</SIGN>
        <UOM>Ea</UOM>
   </QUANTITY>
      <PSCLINENUM>1</PSCLINENUM>
      <USERAREA/>
   </POLINESCHD>
   </POORDERLIN>
   </PROCESS_PO>
  </DATAAREA>
</PROCESS_PO_007>
```

D

Understanding Basic BPEL Process Creation

Overview

To design a composite service, integration developers use a web service composition language BPEL to specify the invocation sequence through Oracle BPEL Process Manager (PM). This composite service has its own WSDL definition and endpoint through the creation of a Partner Link which allows a business event, for example, to be published to the Oracle BPEL Process Manager or to interact with a partner service.

To efficiently utilize a BPEL process in orchestrating a meaningful business flow, the basic concept of creating a BPEL process is discussed in this section.

Understanding BPEL Business Processes

A BPEL process specifies the exact order in which participating web services should be invoked either sequentially or in parallel. In a typical scenario, a BPEL process receives a request. To fulfill it, the process invokes the involved web services and then responds to the original requestor. Because the BPEL process communicates with other Web services, it heavily relies on the WSDL description of the web services invoked by the composite services.

For example, a BPEL process consists of steps that are placed in the exact order that will be invoked. And these steps are called 'activities'. Each activity represents basic construct and is used for a common task, such as use `<invoke>` activity to invoke a web service; use `<reply>` to generate a response for synchronous operations.

Key Activities and Message Patterns

In supporting web services and message exchanges over the web, there are many communication patterns to model the processes. For example, a basic request - response pattern can be used in a synchronous or asynchronous way. A *synchronous request - response* pattern is one that waits for a response before continuing on; while an *asynchronous request - response* pattern does not wait for a response before continuing on

which allows operations to occur in parallel until the response is ready. Another pattern can be *request only* operation that does not require response at all.

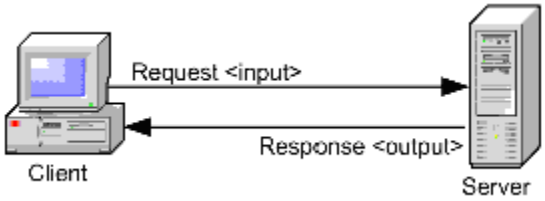
Based on the message patterns, appropriate activities representing various actions can be orchestrated in a meaningful way and enable the services.

To have better explanation about the message patterns and some key activities that are frequently used in a BPEL process, the following two message patterns are used to further describe how to build a simple BPEL process:

Synchronous Request - Response BPEL Process

For synchronous request-response service type, a consumer or client sends a request to a service, and receives an immediate reply.

Synchronous Request - Response Message Pattern

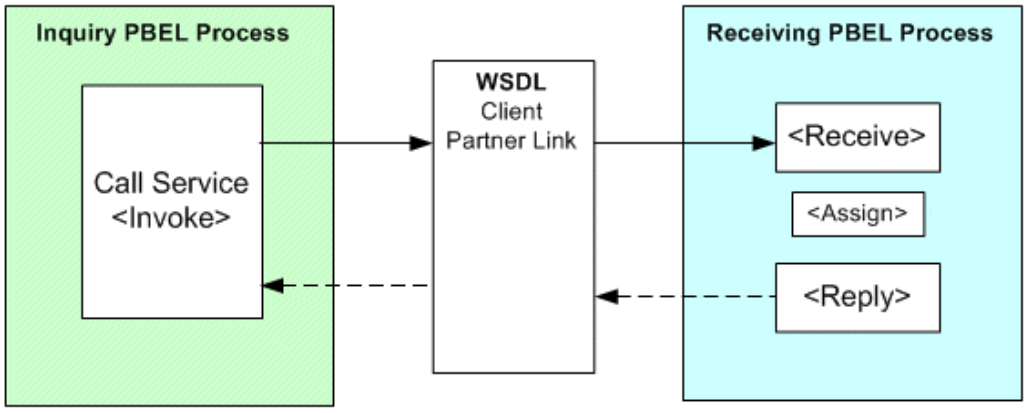


Many online banking tasks are programmed in request-response mode. For example, a request for an account balance is executed as follows:

- A customer (the client) sends a request for an account balance to the Account Record Storage System (the server).
- The Account Record Storage System (the server) sends a reply to the customer (the client), specifying the dollar amount in the designated account.

The synchronous request-response interaction pattern interpreted in a BPEL process can be illustrated in the following diagram:

Synchronous Request-Response Interaction Pattern in BPEL



In the above diagram, there are two BPEL processes involved to complete the synchronous request-response service:

- BPEL Process as Client (Request)

  When the BPEL process is on the client side of a synchronous transaction, it needs an **Invoke** activity to send the request and receive the reply and a **PartnerLink** to carry information between the inquiry BPEL process and a Web service.

- BPEL Process as Service (Response)

  When the BPEL process 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). Additionally, it requires a **PartnerLink** to carry information between the web service and the inquiry BPEL process.

  > **Note:** Sometime, an Assign activity is placed before a Reply activity to take received data as an input variable and assign it the Reply activity as an output variable in respond to the request.

Partner Link

A partner link defines the location and the role of the web services that the BPEL process connects to in order to perform tasks, as well as the variables used to carry information between the web service and the BPEL process. A partner link is required for each web service that the BPEL process calls.

Invoke Activity

An Invoke activity opens a port in the BPEL process to send and receive data. It uses this port to submit the required data and receive the response.

In the account balance inquiry example, the Invoke activity submits the account number

entered by the customer to the server and receives dollar amount in return as the account balance. For synchronous callbacks, the Java rule function supports the feature through Business Event System. Thus, only one port is needed for both the send and receive functions.

Receive Activity

A Receive activity waits for an incoming request data as an input variable. For example, the Receive activity accepts the account balance inquiry by taking the account number as an input variable to the server.

Reply Activity

A Reply activity enables the business process to send a response message in reply to a message that was received through a Receive activity. For example, the Reply activity takes the account balance from the server as an output variable and sends it back to the requestor.

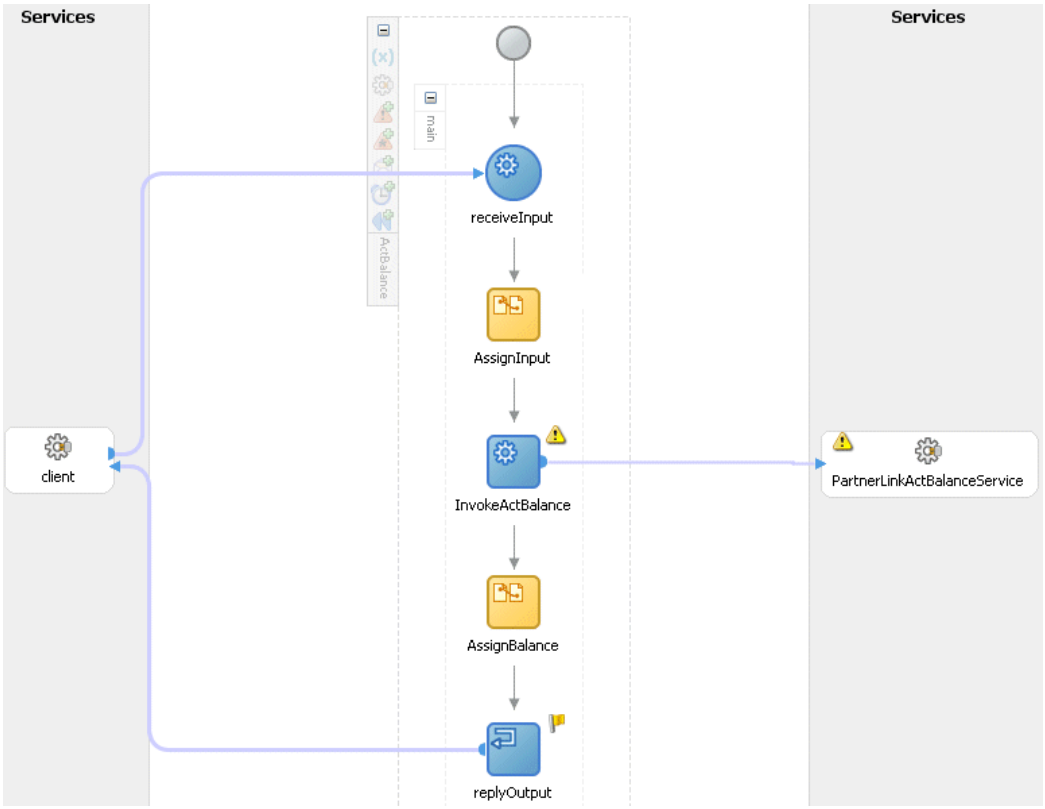
The combination of a receive and a reply forms a request-response operation.

Orchestrating a Synchronous BPEL Process

The composite BPEL process design flow should be orchestrated with necessary BPEL components or activities so as to successfully invoke a synchronous web service.

Use the account balance inquiry as an example. The service invocation sequence can be orchestrated in Oracle JDeveloper BPEL Designer as shown in the following diagram:

An Example of a BPEL Process Diagram



1. A client sends the request by entering account number for the balance inquiry.
2. The Receive activity receives it as an input variable.
3. The Assign activity takes the account number and passes it to the Invoke activity.
4. The Invoke activity submits the account number entered by the client to a `ActBalanceService` Partner Link and receives dollar amount in return as the account balance.
5. The Assign activity then takes the dollar amount as an input variable and passes it to the Reply activity.
6. The Reply activity takes the account balance and replies to the requestor.

The above composite service - BPEL process requires the following tasks at the design time:

- Adding a Partner Link
- Adding an Invoke Activity

- Adding a Receive Activity
- Adding a Reply Activity
- Adding an Assign Activity

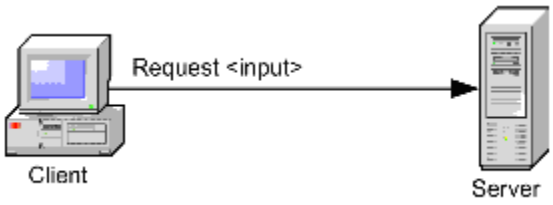
Once a Partner Link is successfully added to a synchronous BPEL project, a WSDL description URL that corresponds to the ActBalanceService business event service with appropriate event payload will be automatically generated. This ActBalanceService Partner Link serves as a bridge to communicate information between the service and the synchronous BPEL project.

> **Note:** The generated WSDL URL of a BPEL process can also be used in defining an event subscription if the BPEL process is used for service invocation through the Business Event System.

One-Way BPEL Process

For One-way or request only service type, there is only one input element which is a client's request for a service and no response is expected.

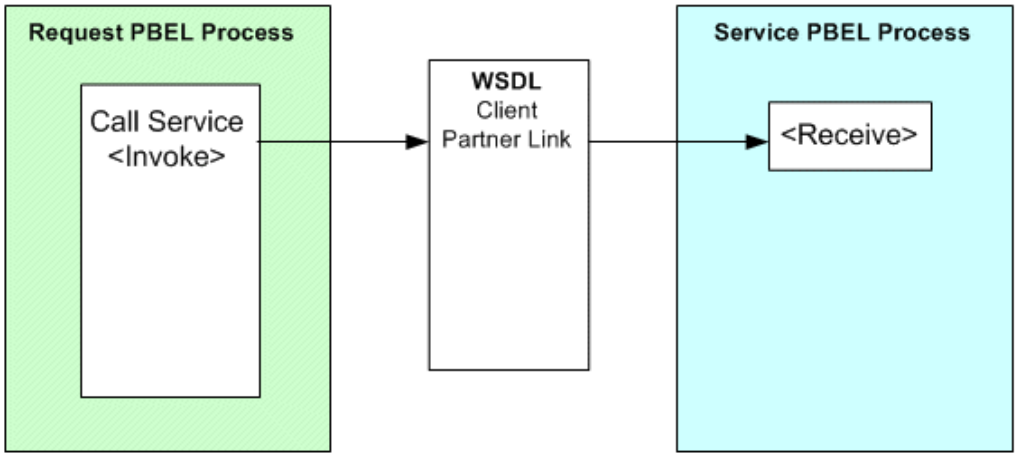
Request Only Message Pattern



For example, a stock symbol sends updated price to the stock quote service when the price change using the request only operation. The server updates the stock price but no response is sent back.

This type of interaction pattern interpreted in a BPEL process can be illustrated in the following diagram:

One-Way (Request Only) Interaction Pattern in BPEL



The following two BPEL processes are involved in this type of service:

- BPEL Process as Client (Request)

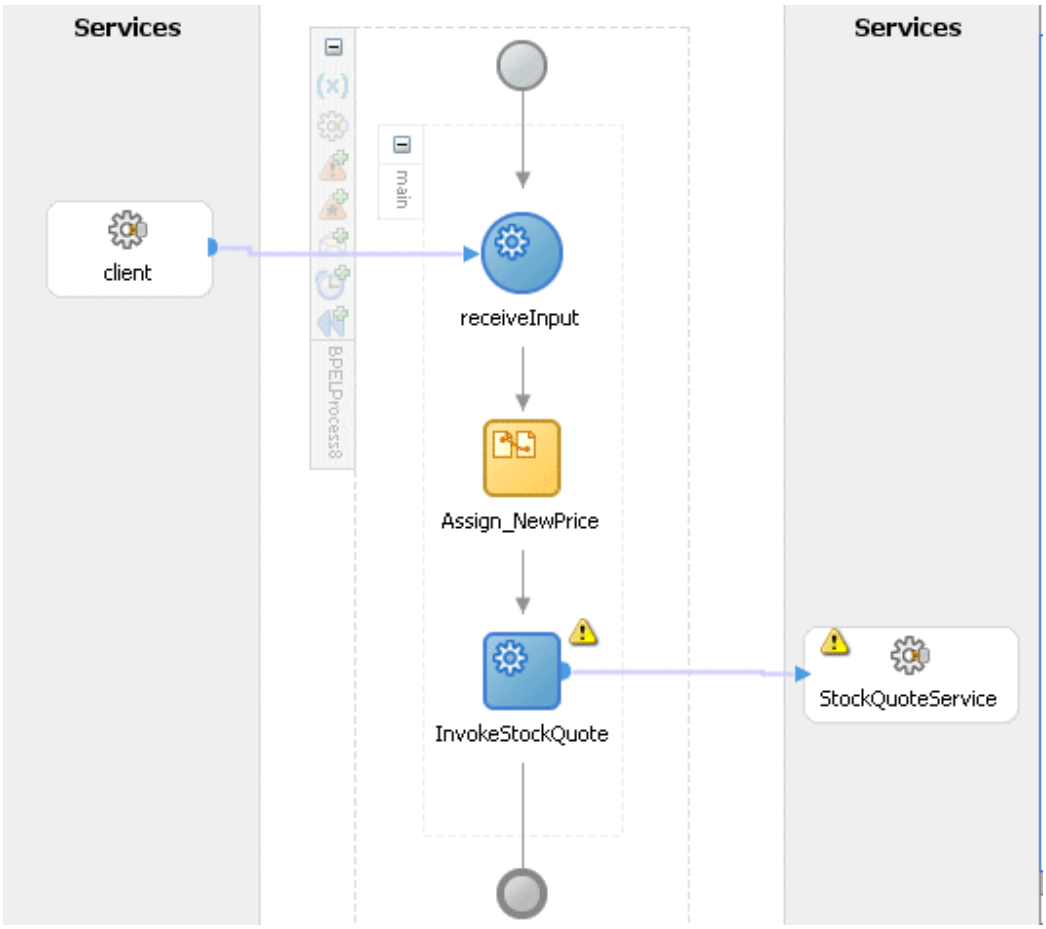
  As the client, the BPEL process needs a valid **PartnerLink** and an **Invoke** activity with the target service and the message. As with all partner activities, the WSDL file defines the interaction.

- BPEL Process as Service

  To accept a message from the client's request, the BPEL process needs a **Receive** activity only.

In the stock quote update example, the Invoke activity submits a stock symbol along with a market price to the StockQuote service in a server. The Receive activity takes the new price as an input and update the StockQuote service.

The following diagram illustrates the service invocation sequence for the stock update example in Oracle JDeveloper BPEL Designer:



1. A client sends the stock symbol and updated price.
2. The Receive activity receives it as an input variable.
3. The Assign activity takes the new price and passes it to the Invoke activity.
4. The Invoke activity submits a stock symbol along with an updated price to the `StockQuoteService` Partner Link in a server.

This composite service - BPEL process requires the following tasks at the design time:

- Adding a Partner Link
- Adding an Invoke Activity
- Adding a Receive Activity
- Adding an Assign Activity

Like synchronous request - response operation, once a Partner Link is successfully added to a BPEL project, a WSDL description URL that corresponds to the service with

appropriate event payload will be automatically generated.

Glossary

Agent

A named point of communication within a system.

Agent Listener

A type of service component that processes event messages on inbound agents.

BPEL

Business Process Execution Language (BPEL) provides a language for the specification of executable and abstract business processes. By doing so, it extends the services interaction model and enables it to support business transactions. BPEL defines an interoperable integration model that should facilitate the expansion of automated process integration in both the intra-corporate and the business-to-business spaces.

Business Event

See Event.

Concurrent Manager

An Oracle E-Business Suite component that manages the queuing of requests and the operation of concurrent programs.

Concurrent Program

A concurrent program is an executable file that performs a specific task, such as posting a journal entry or generating a report.

Event

An occurrence in an internet or intranet application or program that might be significant to other objects in a system or to external agents.

Event Activity

A business event modelled as an activity so that it can be included in a workflow process.

Event Data

A set of additional details describing an event. The event data can be structured as an XML document. Together, the event name, event key, and event data fully communicate what occurred in the event.

Event Key

A string that uniquely identifies an instance of an event. Together, the event name, event key, and event data fully communicate what occurred in the event.

Event Message

A standard Workflow structure for communicating business events, defined by the datatype `WF_EVENT_T`. The event message contains the event data as well as several header properties, including the event name, event key, addressing attributes, and error information.

Event Subscription

A registration indicating that a particular event is significant to a system and specifying the processing to perform when the triggering event occurs. Subscription processing can include calling custom code, sending the event message to a workflow process, or sending the event message to an agent.

Function

A PL/SQL stored procedure that can define business rules, perform automated tasks within an application, or retrieve application information. The stored procedure accepts standard arguments and returns a completion result.

Integration Repository

Oracle Integration Repository is the key component or user interface for Oracle E-Business Suite Integrated SOA Gateway. This centralized repository stores native packaged integration interface definitions and composite services.

Interface Type

Integration interfaces are grouped into different interface types.

JSON

JSON (JavaScript Object Notation) is a text-based open standard designed for human-readable data interchange. The JSON format is often used with REST services to transmit structured data between a server and Web application, serving as an alternative to XML.

Loose Coupling

Loose coupling describes a resilient relationship between two or more systems or organizations with some kind of exchange relationship. Each end of the transaction

makes its requirements explicit and makes few assumptions about the other end.

Lookup Code

An internal name of a value defined in a lookup type.

Lookup Type

A predefined list of values. Each value in a lookup type has an internal and a display name.

Message

The information that is sent by a notification activity. A message must be defined before it can be associated with a notification activity. A message contains a subject, a priority, a body, and possibly one or more message attributes.

Message Attribute

A variable that you define for a particular message to either provide information or prompt for a response when the message is sent in a notification. You can use a predefine item type attribute as a message attribute. Defined as a 'Send' source, a message attribute gets replaced with a runtime value when the message is sent. Defined as a 'Respond' source, a message attribute prompts a user for a response when the message is sent.

Notification

An instance of a message delivered to a user.

Notification Worklist

A Web page that you can access to query and respond to workflow notifications.

Operation

An abstract description of an action supported by a service.

Port

A port defines an individual endpoint by specifying a single address for a binding.

Port Type

A port type is a named set of abstract operations and abstract messages involved.

Process

A set of activities that need to be performed to accomplish a business goal.

REST

Representational State Transfer (REST) is an architecture principle in which the Web services are viewed as resources and can be uniquely identified by their URLs. The key

characteristic of a REST service is the explicit use of HTTP methods (GET, POST, PUT, and DELETE) to denote the invocation of different operations.

SAML Token (Sender-Vouches)

This type of security model authenticates Web services relying on sending a username only through Security Assertion Markup Language (SAML) assertion.

SAML is an XML-based standard for exchanging authentication and authorization data between security domains, that is, between an identity provider and a service provider. SAML Token uses a sender-vouches method to establish the correspondence between a SOAP message and the SAML assertions added to the SOAP message.

See Username Token.

Service

A service is a collection of related endpoints.

Service Component

An instance of a Java program which has been defined according to the Generic Service Component Framework standards so that it can be managed through this framework.

SOA

Service-oriented Architecture (SOA) is an architecture to achieve loose coupling among interacting software components and enable seamless and standards-based integration in a heterogeneous IT ecosystem.

SOAP

Simple Object Access Protocol (SOAP) is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. It uses XML technologies to define an extensible messaging framework providing a message construct that can be exchanged over a variety of underlying protocols.

Subscription

See Event Subscription.

Username Token

A type of security model based on username and password to authenticate SOAP requests at run time.

See SAML Token (Sender-Vouches).

WADL

Web Application Description Language (WADL) is designed to provide a machine-processable description of HTTP-based Web applications. It models the resources provided by a service and the relationships between them.

Web Services

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

Workflow Engine

The Oracle Workflow component that implements a workflow process definition. The Workflow Engine manages the state of all activities for an item, automatically executes functions and sends notifications, maintains a history of completed activities, and detects error conditions and starts error processes. The Workflow Engine is implemented in server PL/SQL and activated when a call to an engine API is made.

WSDL

Web Services Description Language (WSDL) is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint.

WS-Addressing

WS-Addressing is a way of describing the address of the recipient (and sender) of a message, inside the SOAP message itself.

WS-Security

WS-Security defines how to use XML Signature in SOAP to secure message exchanges, as an alternative or extension to using HTTPS to secure the channel.

XML

XML (Extensible Markup Language) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.

Index

E

I

J

O

S

T

U

W

X