

Oracle® E-Business Suite

Integrated SOA Gateway Developer's Guide

Release 12.2

Part No. E20927-23

December 2023

Oracle E-Business Suite Integrated SOA Gateway Developer's Guide, Release 12.2

Part No. E20927-23

Copyright © 2008, 2023, Oracle and/or its affiliates.

Primary Author: Melody Yang

Contributor: Rekha Ayothi, Sudipto Chakraborty, Rajesh Ghosh, Vardhan Kale, Megha Mathpal, Ravindra Nadakuditi, Saritha Nalagandla, Dilbagh Sardar, Vijay Shanmugam, Shivdas Tomar, Abhishek Verma

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

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

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

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

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

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

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>.

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.

Contents

Send Us Your Comments

Preface

1 Oracle E-Business Suite Integrated SOA Gateway Overview

Oracle E-Business Suite Integrated SOA Gateway Overview.....	1-1
Major Components Features and Definitions.....	1-2

2 Discovering and Viewing Integration Interfaces and Services

Overview.....	2-1
Searching and Viewing Integration Interfaces.....	2-1
Reviewing Interface Details.....	2-5
Generating SOAP Web Services.....	2-6
Reviewing WSDL Element Details.....	2-8
Reviewing WADL Element Details.....	2-18
Understanding SOAP Messages.....	2-24
Understanding REST Messages.....	2-43

3 Using PL/SQL APIs as Web Services

Overview.....	3-1
Using PL/SQL SOAP Services	3-2
Invoking a Synchronous Web Service from a SOA Composite Application with BPEL Process.....	3-6
Creating a SOA Composite Application with BPEL Process.....	3-7
Creating a Partner Link for the Web Service.....	3-10
Adding a Partner Link for File Adapter.....	3-11

Adding Invoke Activities.....	3-19
Adding Assign Activities.....	3-23
Deploying and Testing the SOA Composite with Synchronous BPEL Process.....	3-37
Deploying the SOA Composite with BPEL Process.....	3-38
Testing the SOA Composite Application.....	3-42
Invoking an Asynchronous Web Service from a SOA Composite Application with BPEL Process.....	3-46
Creating a SOA Composite Application with BPEL Process.....	3-49
Adding a Partner Link.....	3-51
Adding an Invoke Activity.....	3-52
Adding a Receive Activity.....	3-53
Adding Assign Activities.....	3-55
Deploying and Testing the SOA Composite with Asynchronous BPEL Process.....	3-61
Deploying the SOA Composite with BPEL Process.....	3-62
Testing the SOA Composite Application.....	3-65
Using PL/SQL REST Services.....	3-68
Invoking a REST Service Using HTTP Basic Authentication and XML Payload With REST Header.....	3-69
Deploying a PL/SQL REST Web Service.....	3-70
Recording Resource Information from Deployed WADL.....	3-72
Creating a Project with a Java Class.....	3-73
Invoking a REST Service Using a Java Class.....	3-79
Invoking a REST Service Using Token Based Authentication and JSON Payload.....	3-79
Deploying a PL/SQL REST Web Service.....	3-81
Recording the Deployed WADL URL.....	3-83
Creating a Project with a Java Class.....	3-84
Invoking REST Service Using a Java Client.....	3-95

4 Using Java APIs as REST Services

Overview.....	4-1
Invoking a Java Bean Service Using HTTP GET Method.....	4-2
Deploying a REST Service.....	4-2
Creating a Security Grant.....	4-4
Recording Resource Information from Deployed WADL.....	4-6
Creating a Project with a Java Class.....	4-8
Invoking a REST Service Using a Java Class.....	4-14
Annotating and Invoking a Custom Java Bean Service.....	4-15
Creating and Compiling Custom Java APIs.....	4-17
Deploying Custom Java Classes and Source Files.....	4-34
Parsing and Uploading the Annotated Custom Java Bean Service to the Integration Repository.....	4-34

Deploying a Custom Java Bean Service.....	4-36
Creating a Security Grant.....	4-37
Recording Resource Information from Deployed WADL.....	4-38
Invoking a Custom REST Service from HTML Using Javascript.....	4-41
Invoking an Application Module Service Using Token Based Authentication and XML	
Payload.....	4-48
Deploying a REST Service.....	4-49
Recording the Deployed WADL URL.....	4-51
Creating a Security Grant.....	4-52
Creating a Project with a Java Class.....	4-53
Invoking a REST Service Using a Java Class.....	4-60

5 Using XML Gateway Inbound and Outbound Interfaces

Overview.....	5-1
Using XML Gateway Inbound Services.....	5-2
Using XML Gateway Inbound Services at Design Time.....	5-2
Creating a New SOA Composite Application with BPEL Process.....	5-7
Creating a Partner Link.....	5-11
Adding Partner Links for File Adapter.....	5-12
Adding Invoke Activities.....	5-16
Adding Assign Activities.....	5-17
Deploying and Testing the SOA Composite with BPEL Process at Runtime.....	5-22
Deploying the SOA Composite with BPEL Process.....	5-22
Testing the SOA Composite Application with BPEL Process.....	5-25
Using XML Gateway Outbound Through Subscription Model.....	5-30
Using XML Gateway Outbound Services at Design Time.....	5-30
Creating a New SOA Composite Application with BPEL Process.....	5-34
Creating a Partner Link for AQ Adapter.....	5-35
Adding a Receive Activity.....	5-44
Adding a Partner Link for File Adapter.....	5-46
Adding an Invoke Activity.....	5-48
Adding an Assign Activity.....	5-49
Deploying and Testing the SOA Composite Application with BPEL Process at Runtime	
.....	5-50
Deploying the SOA Composite Application with BPEL Process.....	5-51
Testing the SOA Composite Application with BPEL Process.....	5-53

6 Using Business Events Through Subscription Model

Overview.....	6-1
Using a Business Event in Creating a SOA Composite Application with BPEL Process at	

Design Time	6-2
Creating a New SOA Composite Application with BPEL Process.....	6-3
Creating a Partner Link for AQ Adapter.....	6-5
Adding a Receive Activity.....	6-11
Adding a Partner Link for File Adapter.....	6-13
Adding an Invoke Activity.....	6-19
Adding an Assign Activity.....	6-20
Deploying and Testing the SOA Composite Application with BPEL Process at Runtime ...	6-21
Deploying the SOA Composite Application with BPEL Process	6-22
Testing the SOA Composite Application with BPEL Process.....	6-23

7 Using Concurrent Programs

Overview	7-1
Using Concurrent Program WSDLs at Design Time	7-2
Creating a New SOA Composite Application with BPEL Process.....	7-6
Creating a Partner Link for the Web Service.....	7-7
Adding a Partner Link for File Adapter.....	7-8
Adding Invoke Activities.....	7-13
Adding Assign Activities.....	7-15
Deploying and Testing the SOA Composite with BPEL Process at Runtime	7-21
Deploying the SOA Composite Application with BPEL Process.....	7-22
Testing the SOA Composite Application with BPEL Process.....	7-23

8 Using Open Interface REST Services

Overview	8-1
Using an Open Interface Table REST Service	8-1
Deploying a REST Service.....	8-2
Creating a Security Grant.....	8-4
Recording Resource Information from Deployed WADL.....	8-4
Invoking a REST Service Using Command Lines.....	8-4

9 Using Business Service Objects

Overview	9-1
Using Business Service Object SOAP Services	9-2
Using Business Service Object WSDLs at Design Time.....	9-5
Creating a SOA Composite Application with BPEL Process.....	9-6
Creating a Partner Link.....	9-7
Adding a Partner Link for File Adapter.....	9-8
Adding an Invoke activity.....	9-12
Adding an Assign activity.....	9-15

Deploying and Testing the SOA Composite with BPEL Process at Runtime.....	9-22
Deploying the SOA Composite with BPEL Process.....	9-22
Testing the SOA Composite Application with BPEL Process.....	9-24
Using Business Service Object REST Services.....	9-25
Deploying a Business Service Object REST Service.....	9-25
Creating a Security Grant for the Deployed Service.....	9-27
Recording Resource Information from Deployed WADL.....	9-28
Invoking a REST Service Using Command Lines.....	9-28
10 Using Composite Services - BPEL	
Overview.....	10-1
Viewing Composite Services - BPEL.....	10-2
Downloading Composite Services - BPEL.....	10-2
Modifying and Deploying BPEL Processes.....	10-3
11 Creating and Using Custom Integration Interfaces	
Overview.....	11-1
Creating Custom Integration Interfaces.....	11-2
Creating Custom Integration Interfaces of Native Interface Types.....	11-2
Creating Custom Composite Services - BPEL.....	11-10
Creating Custom Business Events Using Workflow XML Loader.....	11-16
Using Custom Integration Interfaces as Web Services.....	11-24
Using Custom Interface WSDL in Creating a SOA Composite Application with BPEL Process at Design Time.....	11-25
Creating a New SOA Composite Application with BPEL Process.....	11-27
Creating a Partner Link for the Web Service.....	11-29
Adding a Partner Link for File Adapter.....	11-29
Adding Invoke Activities.....	11-33
Adding Assign Activities.....	11-34
Deploying and Testing the SOA Composite with BPEL Process at Runtime.....	11-39
Deploying the SOA Composite with BPEL Process.....	11-39
Testing the SOA Composite Application with BPEL Process.....	11-40
12 Using Service Invocation Framework to Invoke SOAP Services	
SOAP Service Invocation Framework Overview.....	12-1
Understanding SOAP Service Message Patterns.....	12-3
Calling Back to Oracle E-Business Suite With SOAP Service Response.....	12-5
Supporting SOAP Service Security.....	12-7
Understanding SOAP Service Input Message Parts.....	12-7
Understanding SOAP Service Invocation Metadata.....	12-12

Invoke Web Service Wizard.....	12-12
SOAP Service Security.....	12-14
Additional Subscription Parameters.....	12-14
Managing SOAP Service Invocation Errors.....	12-15
Defining SOAP Service Invocation Metadata.....	12-16
Step 1: Creating a SOAP Service Invoker Business Event.....	12-17
Step 2: Creating a Local Subscription to the SOAP Service Invoker Event.....	12-19
Step 3: Creating an Error Event Subscriptions to Enable Error Processing for the SOAP Service.....	12-21
Step 4: Creating a Receive Event for SOAP Response (Optional).....	12-22
Step 5: Creating a Receive Subscription for SOAP Response (Optional).....	12-24
Invoking SOAP Services.....	12-25
Testing SOAP Service Invocation.....	12-29
An Example of Invoking a SOAP Service from a Workflow Process.....	12-33
Troubleshooting SOAP Service Invocation Failure.....	12-35
Extending Seeded Java Rule Function for SOAP Services.....	12-41
Other SOAP Service Invocation Usage Considerations.....	12-48

13 Using Service Invocation Framework to Invoke REST Services

REST Service Invocation Framework Overview.....	13-1
Understanding REST Service Message Patterns.....	13-3
Calling Back to Oracle E-Business Suite with REST Service Response.....	13-3
Supporting REST Service Security.....	13-3
Understanding REST Service Invocation Metadata.....	13-4
REST Service Details.....	13-5
Advanced Configuration.....	13-9
Managing REST Service Invocation Errors.....	13-11
Defining REST Service Invocation Metadata.....	13-11
Step 1: Creating a REST Service Invoker Business Event.....	13-12
Step 2: Creating a Local Subscription to Invoke the REST Service.....	13-13
Step 3: Creating an Error Subscription to Enable Error Processing for the REST Service	13-16
Step 4: Creating a Receive Event for REST Response (Optional).....	13-18
Step 5: Creating a Receive Event Subscription for REST Response (Optional).....	13-19
Invoking REST Services	13-20
Testing REST Service Invocation.....	13-22
An Example of Invoking a REST Service from a Java API.....	13-24
Troubleshooting REST Service Invocation Failure	13-28
Extending Seeded Java Rule Function for REST Services.....	13-29

A Integration Repository Annotation Standards

General Guidelines.....	A-1
Java Annotations.....	A-4
PL/SQL Annotations.....	A-17
Concurrent Program Annotations.....	A-23
XML Gateway Annotations.....	A-25
Business Event Annotations.....	A-35
Business Entity Annotation Guidelines.....	A-41
Composite Service - BPEL Annotation Guidelines.....	A-115
Glossary of Annotations.....	A-120

B Configuring Server Connection

Application Server Connection	B-1
-------------------------------------	-----

C Sample Payload

Sample Payload for Creating Supplier Ship and Debit Request.....	C-1
Sample Payload for Inbound Process Purchase Order XML Transaction.....	C-3

Glossary

Index

Send Us Your Comments

Oracle E-Business Suite Integrated SOA Gateway Developer's Guide, Release 12.2

Part No. E20927-23

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 12.2 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 xiv 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

- 1 Oracle E-Business Suite Integrated SOA Gateway Overview
 - 2 Discovering and Viewing Integration Interfaces and Services
 - 3 Using PL/SQL APIs as Web Services
 - 4 Using Java APIs as REST Services
 - 5 Using XML Gateway Inbound and Outbound Interfaces
 - 6 Using Business Events Through Subscription Model
 - 7 Using Concurrent Programs
 - 8 Using Open Interface REST Services
 - 9 Using Business Service Objects
 - 10 Using Composite Services - BPEL
 - 11 Creating and Using Custom Integration Interfaces
 - 12 Using Service Invocation Framework to Invoke SOAP Services
 - 13 Using Service Invocation Framework to Invoke REST Services
 - A Integration Repository Annotation Standards
 - B Configuring Server Connection
 - C Sample Payload
- Glossary

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 12.2 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/E26401_01/index.htm), provides the latest updates to Oracle E-Business Suite Release 12.2 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 as an application in Oracle E-Business Suite.

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 Cloud Using the Oracle E-Business Suite Adapter with Oracle Integration 3

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.

Note that this book is the latest generation of Oracle Integration, Oracle Integration 3. Its prior generation, Oracle Integration Generation 2 is called *Oracle Cloud Using the Oracle E-Business Suite Adapter with Oracle Integration*. Both books are part of the integration documentation in Oracle Cloud Platform as a Service (PaaS) and are available in the Oracle Cloud Library on the Oracle Help Center.

Oracle E-Business Suite Concepts

This book is intended for all those planning to deploy Oracle E-Business Suite Release 12.2, 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 any installation and configuration choices that are available.

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 Oracle E-Business Suite Development. 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 Electronic Technical Reference Manual User's Guide

This guide describes how to set up and navigate Oracle E-Business Suite Electronic Technical Reference Manual (eTRM) user interface in Oracle E-Business Suite. It also explains how to browse and search the Oracle eTRM repository to locate desired FND and database metadata and objects, and how to view object details, reports, and diagrams.

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 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 set up and implement Service Invocation Framework to invoke SOAP and REST services from Oracle E-Business Suite, and how to manage web service security, configure logs, and monitor both inbound service invocations using Service Monitor and outbound service invocations through Service Invocation Framework using Service Invocation Monitor.

Oracle E-Business Suite Maintenance Guide

This guide explains how to patch an Oracle E-Business Suite system, describing the adop patching utility and providing guidelines and tips for performing typical patching operations. It also describes maintenance strategies and tools designed to help keep a system running smoothly.

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

This guide includes the latest mobile release with new underlying technologies, as well as the earlier mobile releases built with Oracle Mobile Application Framework (MAF). It explains 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. 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 includes information for the latest mobile release with new underlying technologies, as well as the earlier mobile releases built with Oracle Mobile Application Framework (MAF). For mobile releases built with MAF, 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 Security Guide

This guide contains information on a comprehensive range of security-related topics, including access control, user management, function security, data security, secure configuration, and auditing. It also describes how Oracle E-Business Suite can be integrated into a single sign-on environment.

Oracle E-Business Suite User's Guide

This guide explains how to navigate, enter and query data, and run concurrent requests using the user interface (UI) of Oracle E-Business Suite. It includes information on setting preferences and customizing the UI. In addition, this guide describes accessibility features and keyboard shortcuts for Oracle E-Business Suite.

Oracle Fusion Middleware Oracle E-Business Suite Adapter User's Guide

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

This book is available in the Oracle Fusion Middleware 12c Documentation Library and Oracle Fusion Middleware 11g Documentation Library.

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 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 configure message metadata for Oracle Mobile Approvals for Oracle E-Business Suite and how to define and customize business events and event subscriptions.

Oracle Workflow API Reference

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

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.

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, an essential component in ISG, can be transformed into SOAP-based and REST-based web services.

- By leveraging Oracle SOA Suite running on Oracle WebLogic Server, ISG can expose various integration interfaces within Oracle E-Business Suite as SOAP-based web services. These SOAP services described in WSDLs are deployed to Oracle SOA Suite for service consumption.
- Without the dependency on Oracle SOA Suite, REST-based services provided through ISG are developed with the infrastructure of Oracle E-Business Suite. REST services described in WADLs are directly deployed to an Oracle E-Business Suite WebLogic environment. They can be used for user-driven applications such as Oracle E-Business Suite mobile applications.

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

Major Features

Oracle E-Business Suite Integrated SOA Gateway can do the following:

- 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 and asynchronous (callback without acknowledgement only) interaction patterns for SOAP-based web services

Note: In this release, only PL/SQL APIs can be enabled with the support for asynchronous service pattern.

- Support synchronous interaction pattern for REST-based web services

Note: In this release, only PL/SQL APIs, Concurrent Programs, Business Service Objects, Java Bean Services, Application Module Services, Open Interface Tables, and Open Interface Views can be exposed as REST services.
- 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 process administrative functions
- Provide centralized, user-friendly logging configuration for web services generated through the service provider of Oracle E-Business Suite Integrated SOA Gateway
- Audit and monitor Oracle E-Business Suite inbound service operations from Service Monitor
- Audit and monitor outbound service invocations from Oracle E-Business Suite through Service Invocation Monitor
- Leverage Oracle Workflow Business Event System to enable web service invocation from Oracle E-Business Suite

Major Components Features and Definitions

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

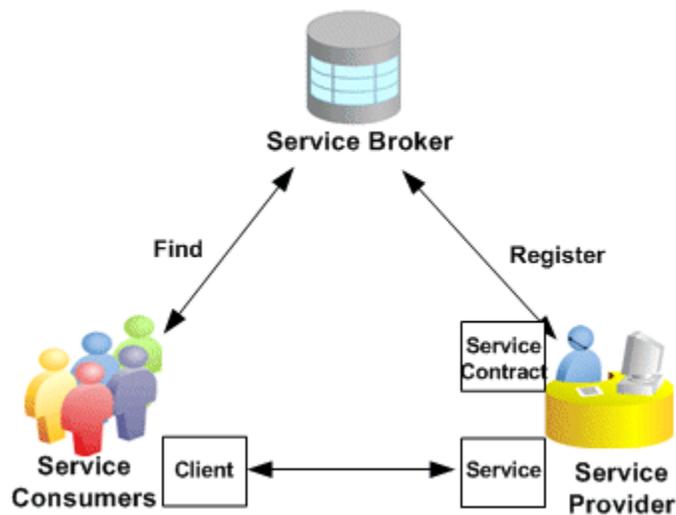
component.

Enabling Oracle E-Business Suite Web Services

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 resided in Oracle Integration Repository to be transformed into web services. SOAP services are deployed from the Integration Repository to Oracle SOA Suite allowing more consumptions over the web. REST services are deployed to Oracle E-Business Suite.

The basic concept of web service components is illustrated in the following diagram:

Web Service Components



- Service Provider is the primary engine underlying the web services. It acts as a bridge between Oracle E-Business Suite and Oracle SOA Suite to facilitate the service enablement for various types of Oracle E-Business Suite interfaces.

Note: In earlier Oracle E-Business Suite Releases, SOA Provider and Web Service Provider were used in enabling Oracle E-Business Suite web services. In the Release 12.2, Service Provider is the engine for service enablement.

Service Provider leverages Oracle SOA Suite for provisioning Oracle E-Business Suite SOAP-based services. It is the engine that performs the actual service generation and deployment behind the scene.

- Service Consumer (Web Service Client) is the party that uses or consumes the services provided by the Service Provider.

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

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. It supports the following interface types:

- PL/SQL
- XML Gateway
- Concurrent Programs
- Business Events
- 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 are not serviceable interfaces and cannot be exposed as SOAP services. Refer to My Oracle Support Knowledge Document 966982.1 for the suggested alternatives to the existing Java APIs for Forms interfaces.

- Composite Interfaces

Oracle E-Business Suite Integrated SOA Gateway leverages Oracle Integration Repository to provide the capabilities of web service generation and deployment, as well as managing the service development life cycle.

Note: Not all the interface types resided in the Integration Repository can be service enabled. The supported interface types for service enablement are XML Gateway, PL/SQL, Concurrent Program, Business Service Object, Java Bean Services, Open Interface Tables, and Open Interface Views.

As mentioned earlier, security services are predeployed REST services from Oracle Application Object Library. There is no need to enable the security services from Integration Repository as required by other supported interface types.

Service Invocation Framework

Service Invocation Framework (SIF) leverages Oracle Workflow Java Business Event System (JBES) and a seeded Java rule function to invoke SOAP and REST services from Oracle E-Business Suite.

It provides an infrastructure allowing developers to interact with SOAP and REST services through service endpoint descriptions.

For more information about how to invoke services using the Service Invocation Framework, see:

- Using Service Invocation Framework to Invoke SOAP Services, page 12-1
- Using Service Invocation Framework to Invoke REST Services, page 13-1

Service Monitor

Service Monitor known as SOA Monitor is a light-weight service monitoring and

management tool.

Service Monitor fetches data and statistics for each instance of a service request and response and lets administrators monitor all *inbound* Oracle E-Business Suite service invocations provided through Oracle E-Business Suite Integrated SOA Gateway. Use Service Monitor to view the runtime request and response data received and sent directly to Oracle E-Business Suite for REST services and the runtime messages passed through Oracle SOA Suite for SOAP services.

For more information about Service Monitor, see *Monitoring and Managing Inbound Service Invocation Messages Using Service Monitor*, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

Service Invocation Monitor

Service Invocation Monitor is also a service monitoring and management tool. It tracks all outbound service invocations from Oracle E-Business Suite through Service Invocation Framework.

For more information about Service Invocation Monitor, see *Monitoring and Managing Outbound Service Invocation Messages Using Service Invocation Monitor*, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

Web Service 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, the request messages must be checked based on the selected authentication type:
 - The SOAP messages must be authenticated using UsernameToken or SAML Token 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 access 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.

Discovering and Viewing Integration Interfaces and Services

Overview

In addition to browsing and viewing integration interfaces from Oracle Integration Repository, users who have the Integration Developer role can generate SOAP web services for selected interfaces.

To better understand these design-time tasks and the structures of WSDL and WADL elements, this chapter includes the following topics:

- Searching and Viewing Integration Interfaces, page 2-1
- Reviewing Interface Details, page 2-5
- Generating SOAP Web Services, page 2-6
- Reviewing WSDL Element Details, page 2-8
- Reviewing WADL Element Details, page 2-18
- Understanding SOAP Messages, page 2-24
- Understanding REST Messages, page 2-43

Searching and Viewing Integration Interfaces

Browsing the Integration Interfaces

There are many ways to locate and view integration interfaces resided in Oracle Integration Repository. You can browse the interfaces by selecting either one of the following selections from the View By drop-down list:

- Product Family
- Interface Type
- Standard

After the selection, expand the tree node 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. Click the **Show More Search Options** link in the Search page to display more search fields. For example, the Scope field has *Private to Application, Internal to Oracle, Public, and All* drop-down values for your selection. If 'All' is selected from the Scope field, then all integration interfaces including public, private to application, and internal to Oracle interfaces will be listed in the results region.

Note: Integration analysts only have 'All' (default) and 'Public' list of values displayed from the Scope drop-down list, and only Public integration interfaces will be retrieved as the search result even if the default value 'All' is selected 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*.

Searching for Deployed Web Services

To locate deployed services for concurrent programs, for example, select 'Concurrent Program' in the Interface field first, then click the **Show More Search Options** link, and then select 'Deployed' in the Web Service Status field. After you perform the search, all deployed concurrent programs are retrieved and shown as the search results..

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.

Search Page for Application Module Services

Integration Repository

Search Browse

Interface Name Internal Name
Product Family All Interface Type Java
Product All Business Entity

[Hide More Search Options](#)
[TIP](#) Select Category before a Category Value

Category Interface Subtype Web Service Type All
Category Value Java Bean Services Web Service Status All
Interface Source All Standard All
Scope All Standard Specification
Status All

| ...

Name ▲	Internal Name ▲	Product ▲	Type ▲	Source ▲	Status ▲	Description
Advanced Scheduler REST API	oracle.apps.csr.webservice.server.SchedulerRestServicesImpl	Scheduler	Java	Oracle	Active	This API provides REST based services for the functionality of Advanced Scheduler.

Copyright (c) 1998, 2019, Oracle and/or its affiliates. All rights reserved. About this Page Privacy Statement

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 Java APIs for Forms interfaces, they are not serviceable interfaces and cannot be exposed as SOAP services. 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 interfaces.

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

Once annotated custom interface definitions are successfully uploaded to the Integration Repository, they are merged into the interface types to which they belong

and displayed together with Oracle seeded interfaces from the Integration Repository browser window. Interface Source 'Custom' is used to categorize those custom integration interfaces, in contrast to Interface Source 'Oracle' for Oracle seeded interfaces in Oracle E-Business Suite.

To locate custom integration interfaces, first click the **Show More Search Options** link to display more search fields.

Search Page for Custom Integration Interfaces

The screenshot shows the 'Integration Repository' search page. The 'Search' section includes fields for Interface Name, Internal Name, Product Family, Product, Interface Type, and Business Entity. Below these are expanded search options including Category, Category Value, Interface Source (with 'Custom' selected), Scope, Status, Web Service Type, Web Service Status, Standard, and Standard Specification. A 'Go' button and 'Clear All' link are present. Below the search filters is a table with columns: Name, Internal Name, Product, Type, Source, Status, and Description. Two results are shown: 'ISG Overload Package' and 'WF Worklist Service'.

Name	Internal Name	Product	Type	Source	Status	Description
ISG Overload Package	ISG_OVERLOAD_PKG	Application Object Library	PL/SQL	Custom	Active	This is a sample Overload plsql Package to calculate area of different shapes
WF Worklist Service	oracle.apps.fnd.wf.worklist.service.rt.server.WFWorklistServiceAMImpl	Application Object Library	Java	Custom	Active	this is a sample WF AM class

Next, select 'Custom' from the Interface Source drop-down list along with any interface type, product family, or scope if needed as the search criteria.

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 E-Business Suite as a user who has the 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 the **Show More Search Options** link to display 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. All integration interfaces including Public, Internal to Oracle, and Private to Application are displayed in the results region.
 7. To view different scopes of integration interfaces, select 'Public', 'Internal to Oracle', or 'Private to Application' from the Scope drop-down list respectively.
 8. Click **Go** to run the search. All interfaces that match your search criteria are displayed.
 9. Select an interface from the search result to view the interface details.

Reviewing Interface Details

To view a selected interface details, click the interface name link that you want to view after performing a search. The interface details page appears allowing you to view the interface general information, full description, source information, and the interface methods (or procedures and functions) region.

Additionally, the web service information if it's available can be displayed in the SOAP Web Service tab, REST Web Service tab, or Web Service region depending on the selected interface type.

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 SOAP or REST web services are deployed, they can be invoked at runtime. For information on how to invoke these services, refer to each individual chapter described

later in this book for details.

Generating SOAP Web Services

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

Selecting Desired Interaction Patterns from the Interaction Pattern Table

SOAP services can be generated with the support for synchronous or asynchronous interaction pattern, or both synchronous and asynchronous patterns to meet your needs.

Before generating a service, an integration administrator or integration developer must select at least one interaction pattern in the Interaction Pattern table for the selected interface or specific methods contained in the interface: This can be achieved at the method level for one or more methods or at the interface level for all methods.

Important: In this release, asynchronous operation is supported only in PL/SQL interfaces in enabling SOAP-based services.

- For XML Gateway and Concurrent Program interface types

Each interface contains only one method and it can only be service enabled synchronously by default; therefore, the Interaction Pattern table is neither displayed in the Web Service region for XML Gateway interfaces nor the SOAP Web Service tab for Concurrent Program interfaces.

- For Business Service Object interface type

Each interface may contain more than one method; therefore, only the Synchronous column is displayed in the Interaction Pattern table for method selection.

By default, none of the interaction pattern would be selected. However, if your system is upgraded from a previous release, for backward compatibility, 'synchronous' pattern is selected for all the methods contained in a service.

Generating Services

In the SOAP Web Service tab (or the Web Service region for an XML Gateway interface), after selecting interaction patterns for an interface, the developer can click **Generate** to generate a web service represented in WSDL with the selected interaction patterns.

SOAP Web Service Tab for a Business Service Object Interface

Overview **SOAP Web Service** REST Web Service Grants

SOAP Service Status Not Generated Design Time Log disabled

Service Operations

Expand All | Collapse All

Display Name	Internal Name	Synchronous	Grant
Account Merge Service	/oracle/apps/ar/hz/service/account/AccountMergeService	<input checked="" type="checkbox"/>	
create Account Merge Request	createAccountMergeRequest	<input checked="" type="checkbox"/>	
get Account Merge Details	getAccountMergeDetails	<input checked="" type="checkbox"/>	

TIP To apply any changes in Interaction Pattern, Generate or Regenerate the service.

* Authentication Type Username Token SAML Token (Sender Vouches)

Generate

After Service Generation

Once a service has been successfully generated, the selected interaction patterns are displayed in the table. Although this table is still editable, any changes to the interaction pattern will be applied only after the service regeneration.

You can expand the interface name node to display all the methods contained in the interface.

After the 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 or the selected interaction pattern information is modified, the web service can be regenerated by clicking **Regenerate**. Upon regeneration, the service definition along with interaction pattern information 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 is exposed as a web service, the corresponding WSDL file is displayed in the interface details page.

Clicking the **View WSDL** link to launch a new window containing the associate WSDL document. This XML-based document describes the selected web service and how the service can be used.

For example, click the deployed **View WSDL** link for the PL/SQL: Invoice Creation from the interface details page, the WSDL document appears.

Deployed SOAP Service WSDL Description

```
infra/services/default/PLSQL_AR_INVOICE_API_PUB/AR_INVOICE_API_PUB_Service?
XSD=xsd/APPS_ISG_CREATE_SINGLE_INVOICE_AR_INVOICE_API_PUB-24CREATE_SIN.xsd" />
</schema>
- <schema xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
targetNamespace="http://xmlns.oracle.com/apps/ar/soapprovider/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 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:OutputParameters" />
- </message>
- <portType name="AR_INVOICE_API_PUB_PortType">
- <operation name="CREATE_INVOICE">
- <input message="tns:CREATE_INVOICE_Input_Msg" />
- <output message="tns:CREATE_INVOICE_Output_Msg" />
- </operation>
- <operation name="CREATE_SINGLE_INVOICE">
- <input message="tns:CREATE_SINGLE_INVOICE_Input_Msg" />
```

Note: The `http://<hostname>:<port>/soa-infra/services/default/<sid>_PLSQL_AR_INVOICE_API_PUB/AR_INVOICE_API_PUB_Service?wsdl` address in the new window has the exact WSDL URL information that appeared in the interface details page. You can copy and use this address directly in any of the web service clients, such as in a BPEL process during a partner link creation.

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.

The *definitions* element often contains an optional `targetNamespace` property, a convention of XML schema that enables the WSDL document to refer to itself.

For example, the *definitions* element in the WSDL document for the Invoice Creation API (AR_INVOICE_API_PUB) appears:

```
<definitions name="AR_INVOICE_API_PUB" targetNamespace="http://xmlns.oracle.com/apps/ar/soapprovider/plsql/ar_invoice_api_pub/" />
```

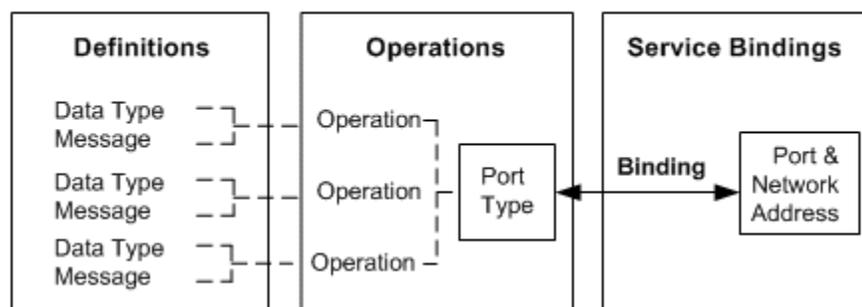
This element specifies numerous namespaces that will be used throughout the remainder of the document.

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.
- **PortType:** It is a set of abstract operations. Each operation refers to an input message and output message.
- **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 illustrates the relationship of the basic parts of WSDL:

WSDL Basic Parts Relationship Diagram



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 Web 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 with the synchronous operation pattern is defined in the APPS_ISG_XX_AR_INVOICE_API_PUB-24CREATE_INV.xsd, and with the asynchronous pattern is defined in the CREATE_INVOICE_ASYNCH.xsd. The message schema location of the CREATE_SINGLE_INVOICE function with the synchronous operation pattern is defined in the APPS_ISG_XX_AR_INVOICE_API_PUB-24CREATE_SIN.xsd, and with the asynchronous pattern is defined in the CREATE_SINGLE_INVOICE_ASYNCH.xsd.

Note: For a generated service that has not yet been deployed, the abstract WSDL description displays a temporary location for the schema location (such as `<include schemaLocation="http://<hostname>:<port>/isgctx/isgapp/plsql/ar_invoice_api_pub/APPS_ISG_XX_AR_INVOICE_API_PUB-24CREATE_INV.xsd" />`).

For a deployed service, a physical location of service endpoint where the service is hosted in soa-infra is displayed instead as shown here:

```

<types>
  <schema elementFormDefault="qualified"
    targetNamespace="http://xmlns.oracle.
com/apps/ar/soapprovider/plsql/ar_invoice_api_pub/create_invoice/">
    <include schemaLocation="http://<soa_suite_hostname>:<port>/soa-
infra/services/default/<jndi_name>_PLSQL_AR_INVOICE_API_PUB/AR_INVOICE_A
PI_PUB_Service?XSD=xsd/APPS_ISG_XX_AR_INVOICE_API_PUB-24CREATE_INV.xsd"
/>
  </schema>
  <schema elementFormDefault="qualified"
    targetNamespace="http://xmlns.oracle.
com/apps/ar/soapprovider/plsql/ar_invoice_api_pub/create_single_invoice/"
>
    <include schemaLocation="http://<soa_suite_hostname>:<port>/soa-
infra/services/default/<jndi_name>_PLSQL_AR_INVOICE_API_PUB/AR_INVOICE_A
PI_PUB_Service?XSD=xsd/APPS_ISG_XX_AR_INVOICE_API_PUB-24CREATE_SIN.xsd"
/>
  </schema>
  <schema elementFormDefault="qualified"
    targetNamespace="http://xmlns.oracle.
com/apps/ar/soapprovider/plsql/ar_invoice_api_pub/">
    <element name="SOAHeader">
      ...
    </element>
  </schema>
  <schema elementFormDefault="qualified"
    targetNamespace="http://xmlns.oracle.
com/apps/ar/soapprovider/plsql/ar_invoice_api_pub/create_invoice/">
    <include schemaLocation="http://<soa_suite_hostname>:<port>/soa-
infra/services/default/<jndi_name>_PLSQL_AR_INVOICE_API_PUB/AR_INVOICE_A
PI_PUB_Service?XSD=xsd/APPS_ISG_XX_AR_INVOICE_API_PUB-24CREATE_INV.xsd"
/>
  </schema>
  <schema elementFormDefault="qualified"
    targetNamespace="http://xmlns.oracle.
com/apps/ar/soapprovider/plsql/ar_invoice_api_pub/create_single_invoice/"
>
    <include schemaLocation="http://<soa_suite_hostname>:<port>/soa-
infra/services/default/<jndi_name>_PLSQL_AR_INVOICE_API_PUB/AR_INVOICE_A
PI_PUB_Service?XSD=xsd/APPS_ISG_XX_AR_INVOICE_API_PUB-24CREATE_SIN.xsd"
/>
  </schema>
  <schema elementFormDefault="qualified"
    targetNamespace="http://xmlns.oracle.
com/apps/ar/soapprovider/plsql/ar_invoice_api_pub/create_invoice_async">
    <include schemaLocation="http://<soa_suite_hostname>:<port>/soa-
infra/services/default/<jndi_name>_PLSQL_AR_INVOICE_API_PUB/AR_INVOICE_A
PI_PUB_Service?XSD=xsd/CREATE_INVOICE_ASYNC.xsd" />
  </schema>
  <schema elementFormDefault="qualified"
    targetNamespace="http://xmlns.oracle.
com/apps/ar/soapprovider/plsql/ar_invoice_api_pub/create_single_invoice_a
synch/">
    <include schemaLocation="http://<soa_suite_hostname>:<port>/soa-
infra/services/default/<jndi_name>_PLSQL_AR_INVOICE_API_PUB/AR_INVOICE_A
PI_PUB_Service?XSD=xsd/CREATE_SINGLE_INVOICE_ASYNC.xsd" />
  </schema>
</types>

```

In addition to message schema locations and schema elements that help to define web messages, the *Types* element can take 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 to set the application context during the service

invocation.

```
...
<schema elementFormDefault="qualified"
  targetNamespace="http://xmlns.oracle.
com/apps/ar/soapprovider/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>
...
```

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 describe 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>
<message name="CREATE_INVOICE_ASYNC_Input_Msg">
  <part name="header" element="tns:SOAHeader" />
  <part name="body" element="tns1:InputParameters" />
</message>
<message name="CREATE_SINGLE_INVOICE_ASYNC_Input_Msg">
  <part name="header" element="tns:SOAHeader" />
  <part name="body" element="tns2:InputParameters" />
</message>
<message name="CREATE_INVOICE_ASYNC_Output_Msg">
  <part name="body" element="tns1:OutputParameters" />
</message>
<message name="CREATE_SINGLE_INVOICE_ASYNC_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 is defined by **CREATE_INVOICE_Input_Msg** for the synchronous operation pattern and **CREATE_INVOICE_ASYNC_Input_Msg** for the asynchronous pattern.

The output message of this function which gives its result is defined by **CREATE_INVOICE_Output_Msg** for the synchronous operation pattern and **CREATE_INVOICE_ASYNC_Output_Msg** for the asynchronous pattern.

The schema of input and output messages is defined in the **APPS_ISG_XX_AR_INVOICE_API_PUB-24CREATE_INV.xsd** for the synchronous pattern, and in the **CREATE_INVOICE_ASYNC.xsd** for the asynchronous pattern.

- **CREATE_SINGLE_INVOICE**

The input message of this function is defined by **CREATE_SINGLE_INVOICE_Input_Msg** for the synchronous operation pattern and **CREATE_SINGLE_INVOICE_ASYNC_Input_Msg** for the asynchronous pattern.

The output message of this function which gives its result is defined by **CREATE_SINGLE_INVOICE_Output_Msg** for the synchronous operation pattern and **CREATE_SINGLE_INVOICE_ASYNC_Output_Msg** for the asynchronous pattern.

The schema of input and output messages is defined in the APPS_ISG_XX_AR_INVOICE_API_PUB-24CREATE_SIN.xsd for the synchronous pattern, and in the CREATE_SINGLE_INVOICE_ASYNC.xsd for the asynchronous pattern.

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 web service authorization.

For more information, see Understanding Web Service Input Message Parts, page 12-7

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* element can combine one request (input message element) message and one response (output message element) message into a Synchronous Request - Response operation, 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:porttype>
</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 Web service example, corresponding to the 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

For the Synchronous Request-Response operation pattern, input and output messages appear as follows:

```
<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>
</portType>
```

For the Asynchronous Request-Response operation pattern, to distinguish it from the rest of operations generated synchronously, **ASYNCH** appears in both input and output (response) messages.

For example, if the 'CREATE_INVOICE' operation is generated asynchronously, ASYNCH appears specifically for the 'CREATE_INVOICE' operation in both input and output (response) messages.

```
...
<portType name="AR_INVOICE_API_PUB_PortType">
  <operation name="CREATE_INVOICE_**ASYNCH**">
    <input name="tns:CREATE_INVOICE_Input_Msg" />
  </operation>
</portType>
<portType name="AR_INVOICE_API_PUB_Callback_PortType">
  <operation name="CREATE_INVOICE_**ASYNCH_RESPONSE**">
    <input name="tns:CREATE_INVOICE_Output_Msg" />
  </operation>
</portType>
<portType name="AR_INVOICE_API_PUB_CallBack_PortType">
  <operation name="CREATE_INVOICE_**ASYNCH_RESPONSE**">
    <input message="tns:CREATE_INVOICE_**ASYNCH**_Output_Msg" />
  </operation>
  <operation name="CREATE_SINGLE_INVOICE_**ASYNCH_RESPONSE**">
    <input message="tns:CREATE_SINGLE_INVOICE_**ASYNCH**_Output_Msg" />
  </operation>
</portType>
```

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.

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"/>
  <wsp:PolicyReference URI="#wss_username_token_service_policy" wsdl:
required="false"/>
  <operation name="CREATE_INVOICE">
    <soap:operation soapAction="CREATE_INVOICE" />
    <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="CREATE_SINGLE_INVOICE" />
    <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>
  <operation name="CREATE_INVOICE_ASYNC">
    <soap:operation soapAction="CREATE_INVOICE_ASYNC" />
    <input>
      <soap:header message="tns:CREATE_INVOICE_ASYNC_Input_Msg" part="
header" use="literal"/>
      <soap:body parts="body" use="literal" />
    </input>
  </operation>
  <operation name="CREATE_SINGLE_INVOICE_ASYNC">
    <soap:operation soapAction="CREATE_SINGLE_INVOICE_ASYNC" />
    <input>
      <soap:header message="tns:CREATE_SINGLE_INVOICE_ASYNC_Input_Msg"
part="header" use="literal"/>
      <soap:body parts="body" use="literal" />
    </input>
  </operation>
</binding>

```

For Asynchronous Request-Response operation pattern, the binding element as shown below describes the SOAP binding for PortType AR_INVOICE_API_PUB_Callback_PortType:

```

<binding name="AR_INVOICE_API_PUB_Callback_Binding" type="tns:
AR_INVOICE_API_PUB_Callback_PortType">
  <soap:binding style="document" transport="http://schemas.xmlsoap.
org/soap/http"/>
    <operation name="CREATE_INVOICE_ASYNCH_RESPONSE">
      <soap:operation soapAction="CREATE_INVOICE_ASYNCH_RESPONSE"/>
      <input>
        <soap:body use="literal" namespace="http://xmlns.oracle.
com/apps/ar/soapprovider/plsql/ar_invoice_api_pub/CREATE_INVOICE_ASYNCH
"/>
      </input>
    </operation>
    <operation name="CREATE_SINGLE_INVOICE_ASYNCH_RESPONSE">
      <soap:operation soapAction="CREATE_SINGLE_INVOICE_ASYNCH_RESPONSE"
/>
      <input>
        <soap:body use="literal" namespace="http://xmlns.oracle.
com/apps/ar/soapprovider/plsql/ar_invoice_api_pub/CREATE_SINGLE_INVOICE_
ASYNCH"/>
      </input>
    </operation>
  </binding>

```

The binding is always document style for SOAP over HTTP binding. It also defines the content of 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.

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 is 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. It 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>

```

For a deployed service, the *Service* element `AR_INVOICE_API_PUB_Service` defines a physical location of service endpoint where the service is hosted in `soa-infra` 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="http://<soa_suite_hostname>:<port>/soa-
infra/services/default/<jndi_name>_PLSQL_AR_INVOICE_API_PUB/AR_INVOICE_A
PI_PUB_Service/" />
  </port>
</service>

```

Note: For a generated service that has not yet been deployed, 'Not_Deployed' is shown in the `soap:address location` element (such as `<soap:address location="#NOT_DEPLOYED#" />`).

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 a concurrent program interface called "Transaction Layout Definition" (ECRDTLD) as an example to explain the WADL elements. 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 the implementation 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/ec/soapprovider/concurrentprogram/rest/ecrdtld/" xmlns="http:
//wadl.dev.java.net/2009/02" xmlns:xsd="http://www.w3.
org/2001/XMLSchema"
xmlns:tns1="http://xmlns.oracle.
com/apps/ec/concurrentprogram/rest/ec/ecrdtld/" name="ECRDTLD"
targetNamespace="http://xmlns.oracle.
com/apps/ec/soapprovider/concurrentprogram/rest/ecrdtld/">
  <grammars>
  ...
  </grammars>
  <resources base="http://<hostname>:<port>/webservicess/rest/ec/">
  ...
  </resources>
</application>

```

Grammars

This element acts as a container for definitions of data exchanged during the implementation of the protocol described by the WADL document. The *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="http:
//<hostname>:<port>/webservicess/rest/ec?XSD=ECRDTLD_SYNCH_TYPEDEF.xsd"
/>
  </grammars>

```

In this example, the included service operation with the support for synchronous pattern ECRDTLD_SYNCH_TYPEDEF is referenced in the *Include* element.

ec 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 (**ec**) 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 *ecrdtld*) contained in the selected interface.

```

<resources base="http://<hostname>:<port>/webservicess/rest/ec/">
  <resource path="/ecrdtld/">
  ...
  </resource>
</resources>

```

Resource

The *resources* element may contain a set of *resource* elements; each resource element represents a REST service operation. In this example, "ecrdtld" is included a 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 Concurrent Program REST services; POST and GET are the supported methods for PL/SQL APIs, 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/ec/">
  <resource path="/ecrdtld/">
    <method id="ECRDTLD" name="POST">
      <request>
        <representation mediaType="application/xml" type="
ECRDTLD_InputParameters" />
        <representation mediaType="application/json" type="
ECRDTLD_InputParameters" />
      </request>
      <response>
        <representation mediaType="application/xml" type="
ECRDTLD_OutputParameters" />
        <representation mediaType="application/json" type="
ECRDTLD_OutputParameters" />
      </response>
    </method>
  </resource>
</resources>
```

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

If the deployed REST service is an interface type of PL/SQL, Business Service Object, 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. The `addGrant` method is implemented with the POST HTTP method, and the `getOperations` method is assisted with the 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 invocation:

Note: Path variables and context parameters are applicable only for the HTTP GET method.

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

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

The 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 runtime 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 the 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 the Java API source file. For the annotation guidelines on Java Bean Services, see Annotations for Java Bean Services, page A-6. For more Application Module Services annotation guidelines, 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 the `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 the 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>

```

- 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 methods. 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 the open interface view contains only one resource entry with the 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, a 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.

SOAP Message Structure

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 can be used to attach security related information targeted at a specific recipient.

For more information, see SOAP Security Header, page 2-26.

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

For more information, see SOAP Header for Application Context, page 2-30.

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

For more information, see SOA Header for XML Gateway Messages, page 2-33.

- Body

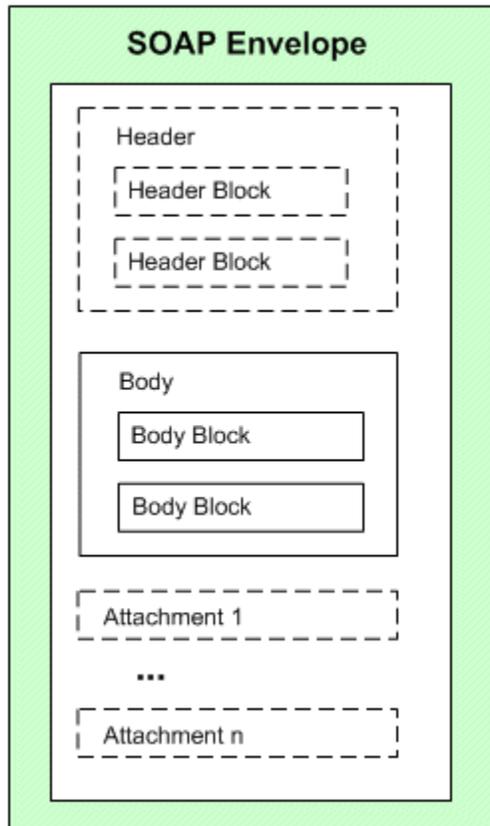
Every envelope element must contain exactly one Body element that holds the message. Immediate child elements of the Body element are called Body Blocks or Parts.

- 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 Oracle SOA Suite for the deployed SOA Composites in an Oracle WebLogic managed server, the SOAP message is

authenticated by a JAAS (Java Authentication and Authorization Service) based login module for Oracle E-Business Suite.

A SOAP message is authenticated using either the UsernameToken or SAML Token security model which has been identified earlier for a service before being deployed. The selected authentication information is embedded in the `wsse:security` Web Security header.

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 the `<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, page 2-40.

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 includes UsernameToken profile which provides user name and password information in the web service security header. User name 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).

Note: The PasswordText password type is the password written in

clear text. SOAP requests invoking the web services should include the security header consisting of a user name and plain text password. The password received as part of the SOAP request at runtime will be validated against the encrypted password stored in Oracle E-Business Suite. After validation, the plain text password from the SOAP request will be discarded.

The user name and password in the SOAP Header of a SOAP message will be passed to authenticate web services. The user name and password discussed here in `wsse:security` is the Oracle E-Business Suite user name and password (or the user name and 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.

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

For detailed instructions on how to pass the security header when invoking an Oracle E-Business Suite web service, see *Configuring Web Service Policies*, page 3-34.

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 Oracle SOA Suite and passed on to a JAAS based login module for Oracle E-Business Suite for authentication based on the `wsse:security` Web Security headers. As part of the validation and processing of the assertions, the receiver or login module for Oracle E-Business Suite must establish the relationship among 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 that 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/soapprovider/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-xxxxxxx" 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-xxxxxxx">
            <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/xxxxxxxxxxxxxxxxxxxxxxxx/xx</ds:DigestValue>
        </ds:Reference>
      </ds:SignedInfo>
      <ds:SignatureValue>

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxx/
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
  </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"
    >ADoNKKuduSTKTwI7jqEzCxD7JU=</wsse:KeyIdentifier></wsse:
SecurityTokenReference>
    </ds:KeyInfo></ds:Signature>
    <Assertion AssertionID="xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
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: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-xxxxxxx" 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/soapprovider/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 (for SSO users) or the Oracle E-Business Suite `FND_USER` table (for non-SSO users).

The SAML assertion in the above SOAP message is for a non-SSO enabled environment. If the user name in the `NameIdentifier` tag is of the form of LDAP DN as shown below, then the user name is verified in the registered OID for SSO users.

```

<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 the 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 Application Context

Application context contains many crucial elements that are used to pass values that may be required in Oracle E-Business Suite. 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.

Application 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 invocation 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 printed in bold:

```
<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>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><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>
```

Application Context in ServiceBean_Header Part of a SOAP Request

These context header elements defined in ServiceBean_Header part of a SOAP request for a Business Service Object service 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 a Business Service Object service.

If the NLS Language element is specified (such as AMERICAN), 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)**
It is an optional parameter to be passed in *ServiceBean_Header* part of a SOAP request for a Business Service Object service.

If a service invocation 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 printed in bold for a business service object service:

```

<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>

```

SOA Header for XML Gateway Messages

In Oracle XML Gateway, each trading partner is configured with Oracle E-Business Suite users. Only these authorized users defined in the Trading Partner Setup form are allowed to perform XML transactions. External clients can pass such usernames in the <USERNAME> and <PASSWORD> elements defined within the <ECX:SOAHeader> element (or <XMLGateway_Header> element for generic XML Gateway services) in the SOAP body. These username parameters are validated by Oracle XML Gateway against the username defined in the trading partner setup before initiating a transaction.

Therefore, for XML Gateway interface type, the authorization check is performed at both the trading partner level, as well as on the username passed in the wsse: security header in the SOAP request. For information on trading partner setup and how to associate users with trading partners, see *Oracle XML Gateway User's Guide*.

The following code snippet shows the SOAHeader element within a SOAP request for an XML Gateway inbound message:

```

<soapenv: Envelope xmlns:ecx="http://xmlns.oracle.
com/apps/ecx/soapprovider/xmlgateway/ecx__cbodi/"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:sys="http://xmlns.oracle.com/xdbsystem">
  <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>

```

XML Gateway header parameters defined in the SOAHeader element (or XMLGateway_Header element for generic XML Gateway services) within a SOAP request are described in the following table:

XML Gateway Header Information in SOAHeader Element within 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	<p>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.</p> <p>Note: For more information, see <i>Destination Trading Partner Location Code</i> in the <i>Oracle XML Gateway User's Guide</i>.</p>
ATTRIBUTE4	This parameter may be defined by the base application.
ATTRIBUTE5	This parameter may be defined by the base application.

Note: The PARTYID and PARTY_TYPE parameters are not used.

The following code snippet shows the XMLGateway_Header element within a SOAP request for a generic XML Gateway service:

```
<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:

XML Gateway 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.
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

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

For information about synchronous SOAP messages, see:

- A Sample Synchronous SOAP Request, page 2-38
- A Sample Synchronous SOAP Response, page 2-40

- A Sample Fault Synchronous SOAP Response, page 2-40

For information about asynchronous SOAP messages, see:

- A Sample Asynchronous SOAP Request, page 2-40
- A Sample Asynchronous SOAP Response, page 2-42
- A Sample Fault Asynchronous SOAP Response, page 2-43

A Sample Synchronous SOAP Request

The following example shows a synchronous 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/soapprovider/plsql/ozf_sd_request_pub/"
xmlns:cre="http://xmlns.oracle.
com/apps/ozf/soapprovider/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 Synchronous SOAP Response

The following example shows a synchronous 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/soapprovider/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 Synchronous SOAP Response

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

For example, the following fault synchronous 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>
```

A Sample Asynchronous SOAP Request

The following example shows an asynchronous SOAP request for CREATE_SD_REQUEST_ASYNCH operation contained in a PL/SQL service OZF_SD_REQUEST_PUB:

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:crel="http://xmlns.oracle.com/isg/ozf_sd_request_pub/CREATE_SD_REQUEST_ASYNC"
  xmlns:ozf="http://xmlns.oracle.com/apps/ozf/soaprovider/plsql/ozf_sd_request_pub/create_sd_request/"
  xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
  <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-3"
        xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
        <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:Nonce EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary">JtcbwUGEcUyy09YgwaPzSA==</wsse:Nonce>
        <wsu:Created>2011-09-21T08:17:10.656Z</wsu:Created>
      </wsse:UsernameToken>
    </wsse:Security>
    <ozf:SOAHeader>
      <!--Optional:-->
      <ozf:Responsibility>OZF_USER</ozf:Responsibility>
      <!--Optional:-->
      <ozf:RespApplication>OZF</ozf:RespApplication>
      <!--Optional:-->
      <ozf:SecurityGroupE>STANDARD</ozf:SecurityGroup>
      <!--Optional:-->
      <ozf:NLSLanguage>AMERICAN</ozf:NLSLanguage>
      <!--Optional:-->
      <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>
        <crel:REQUEST_NUMBER>SDR-CREATE-BPEL001</crel:REQUEST_NUMBER>
        <crel:REQUEST_START_DATE>2011-08-18T12:00:00</crel:REQUEST_START_DATE>
        <crel:REQUEST_END_DATE>2012-10-18T12:00:00</crel:REQUEST_END_DATE>
        <crel:USER_STATUS_ID>1712</crel:USER_STATUS_ID>
        <crel:REQUEST_OUTCOME>IN_PROGRESS</crel:REQUEST_OUTCOME>
        <crel:REQUEST_CURRENCY_CODE>USD</crel:REQUEST_CURRENCY_CODE>
        <crel:SUPPLIER_ID>1718</crel:SUPPLIER_ID>
        <crel:SUPPLIER_SITE_ID>2854</crel:SUPPLIER_SITE_ID>
        <crel:REQUESTOR_ID>100001499</crel:REQUESTOR_ID>
        <crel:ASSIGNEE_RESOURCE_ID>100001499</crel:ASSIGNEE_RESOURCE_ID>
        <crel:ORG_ID>204</crel:ORG_ID>
        <crel:ACCRUAL_TYPE>SUPPLIER</crel:ACCRUAL_TYPE>
        <crel:REQUEST_DESCRIPTION>Create</crel:REQUEST_DESCRIPTION>
        <crel:SUPPLIER_CONTACT_EMAIL_ADDRESS>sdr.supplier@example.com</crel:SUPPLIER_CONTACT_EMAIL_ADDRESS>
        <crel:SUPPLIER_CONTACT_PHONE_NUMBER>2255</crel:SUPPLIER_CONTACT_PHONE_NUMBER>
        <crel:REQUEST_TYPE_SETUP_ID>400</crel:REQUEST_TYPE_SETUP_ID>
        <crel:REQUEST_BASIS>Y</crel:REQUEST_BASIS>
        <crel:USER_ID>1002795</crel:USER_ID>
      </cre:P_SDR_HDR_REC>
    </cre:InputParameters>
  </soapenv:Body>
</soapenv:Envelope>

```

```

<cre:P_SDR_LINES_TBL>
  <crel:P_SDR_LINES_TBL_ITEM>
    <crel:PRODUCT_CONTEXT>PRODUCT</cre:PRODUCT_CONTEXT>
    <crel:INVENTORY_ITEM_ID>149</crel:INVENTORY_ITEM_ID>
    <crel:ITEM_UOM>Ea</crel:ITEM_UOM>
    <crel:REQUESTED_DISCOUNT_TYPE>%</crel:REQUESTED_DISCOUNT_TYPE>
    <crel:REQUESTED_DISCOUNT_VALUE>15.5</crel:REQUESTED_DISCOUNT_VALUE>
    <!--<crel:COST_BASIS>200</crel:COST_BASIS>-->
    <crel:MAX_QTY>200</crel:MAX_QTY>
    <crel:DESIGN_WIN>200</crel:DESIGN_WIN>
    <crel:APPROVED_DISCOUNT_TYPE>%</crel:APPROVED_DISCOUNT_TYPE>
    <crel:APPROVED_DISCOUNT_VALUE>15.5</crel:APPROVED_DISCOUNT_VALUE>
    <crel:APPROVED_MAX_QTY>200</crel:APPROVED_MAX_QTY>
    <crel:VENDOR_APPROVED_FLAG>Y</crel:VENDOR_APPROVED_FLAG>
    <crel:PRODUCT_COST_CURRENCY>USD</crel:PRODUCT_COST_CURRENCY>
    <crel:END_CUSTOMER_CURRENCY>USD</crel:END_CUSTOMER_CURRENCY>
  </crel:P_SDR_LINES_TBL_ITEM>
</cre:P_SDR_LINES_TBL>
<cre:P_SDR_CUST_TBL>
  <crel:P_SDR_CUST_TBL_ITEM>
    <crel:CUST_ACCOUNT_ID>1290</crel:CUST_ACCOUNT_ID>
    <crel:PARTY_ID>1290</crel:PARTY_ID>
    <crel:SITE_USE_ID>10479</crel:SITE_USE_ID>
    <crel:CUST_USAGE_CODE>BILL_TO</crel:CUST_USAGE_CODE>
    <crel:END_CUSTOMER_FLAG>N</crel:END_CUSTOMER_FLAG>
  </crel:P_SDR_CUST_TBL_ITEM>
  <crel:P_SDR_CUST_TBL_ITEM>
    <crel:CUST_ACCOUNT_ID>1287</crel:CUST_ACCOUNT_ID>
    <crel:PARTY_ID>1287</crel:PARTY_ID>
    <crel:SITE_USE_ID>1418</crel:SITE_USE_ID>
    <crel:CUST_USAGE_CODE>CUSTOMER</crel:CUST_USAGE_CODE>
    <crel:END_CUSTOMER_FLAG>Y</crel:END_CUSTOMER_FLAG>
  </crel:P_SDR_CUST_TBL_ITEM>
</cre:P_SDR_CUST_TBL>
</cre:InputParameters>>
</soapenv:Body>
</soapenv:Envelope>

```

A Sample Asynchronous SOAP Response

The following example shows asynchronous response for CREATE_SD_REQUEST_ASYNCH operation contained in a PL/SQL service OZF_SD_REQUEST_PUB:

```

<?xml version="1.0" encoding="UTF-8" ?>
<outputParameters
xmlns:client="http://xmlns.oracle.
com/isg/ozf_sd_request_pub/CREATE_SD_REQUEST_ASYNCH"
xmlns:wsa="http://www.w3.org/2005/08/addressing"
xmlns="http://xmlns.oracle.
com/isg/ozf_sd_request_pub/CREATE_SD_REQUEST_ASYNCH">
  <OutputParameters
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
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"/>
    <X_REQUEST_HEADER_ID>31</X_REQUEST_HEADER_ID>
  </OutputParameters>
</outputParameters>

```

A Sample Fault Asynchronous SOAP Response

For example, the following sample shows the asynchronous response message for incorrect header from soapUI:

```
<?xml version="1.0" encoding="UTF-8" ?>
<outputParameters
xmlns:client="http://xmlns.oracle.
com/isg/ozf_sd_request_pub/CREATE_SD_REQUEST_ASYNCH"
xmlns:wsa="http://www.w3.org/2005/08/addressing"
xmlns="http://xmlns.oracle.
com/isg/ozf_sd_request_pub/CREATE_SD_REQUEST_ASYNCH">
<OutputParameters
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.oracle.
com/apps/ozf/soapprovider/plsql/ozf_sd_request_pub/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_ID xsi:nil="true"/>
</OutputParameters>
</outputParameters>
```

Understanding REST Messages

Based on REST architecture, the REST message uses HTTP header and supported methods to create, update, fetch, or delete 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 2-44.

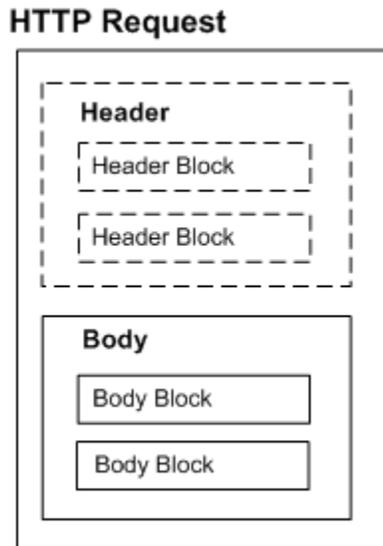
- **Body**

This element defines the main messages or resources.

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

For more information on setting application context, see REST Header for Application Context, page 2-45.

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 the application 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 the application context required for the REST service to be invoked can be initialized, the service can be invoked.

For more information on setting application context, see REST Header for Application Context, page 2-45.

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

REST Header for Application Context

Some Oracle E-Business Suite APIs require application 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>
```

Optional Language Parameters

Similar to the `<NLSLanguage>` parameter to set the language preference, an ISO language parameter `<Language>` provides an alternative way to send the language preference in RFC 5646 format in the REST Header to set the session language, in case web service clients are unaware of the `<NLSLanguage>` parameter used in Oracle E-Business Suite.

Note: The `<Language>` parameter is used for REST services only.

For example, German language in standard RFC 5646 format is `de-DE` or `de`. Set the language to "German" through the `<Language>` parameter:

`<Language>de-DE</Language>` or `<Language>de</Language>`

Please note that the `<NLSLanguage>` and `<Language>` parameters are both optional. The `<NLSLanguage>` parameter takes precedence over the `<Language>` parameter if the `<NLSLanguage>` parameter value (such as `<NLSLanguage>GERMAN</NLSLanguage>`) is passed and valid.

- If the `<NLSLanguage>` value is not provided, then:
 - If the `<Language>` parameter is passed with a valid value, then the `<Language>` value will be used.
 - If the `<Language>` parameter is passed with an invalid value, then an error message would be returned.

For example, if `de-DE` is passed as the `<Language>` value, but German is not an enabled or installed Oracle E-Business Suite language, then an error appears indicating that the language is not installed.

- If the `<NLSLanguage>` value is invalid or not installed, then an error message would be returned.

Only when both `<NLSLanguage>` and `<Language>` values are not provided, the `Accept-Language` header (such as `Accept-Language = de-DE`) from the HTTP Header will be considered. If the `Accept-Language` header is not passed or the passed value is invalid or not installed, then user's default language will be used.

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/soapprovider/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/soapprovider/plsql/rest/fnd_user_pkg/">
<grammars>
  <include xmlns="http://www.w3.org/2001/XMLSchema" href="https:
//<hostname>:<port>/webservicess/rest/FndUserSvc/?
XSD=TESTUSERNAME_SYNCH_TYPEDEF.xsd" />
</grammars><resources base="http://<hostname>:
<port>/webservicess/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>/webservicess/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>/webservicess/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 the XML and JSON formats are listed in the following table:

REST Messages with the XML and JSON Formats

Input Payload (Request Message)	<p>XML-based REST Message</p> <pre><?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></pre> <p>JSON-based REST Message</p> <pre>{ "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" } }}</pre>
Response	<p>XML-based REST Message</p> <pre><?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></pre> <p>JSON-based REST Message</p> <pre>{ "OutputParameters" : { "@xmlns:xsi" : "http://www.w3.org/2001/XMLSchema- instance" , "@xmlns" : "http://xmlns.oracle. com/apps/fnd/rest/fndGlobalSvc/user_id/" , "TESTUSERNAME" : "2" } }</pre>

REST Messages with the XML and JSON Formats

Error Response	<p>XML-based REST Message</p> <pre><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></pre> <p>JSON-based REST Message</p> <pre>{ "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" } } }</pre>
----------------	---

For more examples of REST messages used in OZF_SD_REQUEST_PUB service invocation, see Examples of REST Messages, page 2-49.

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 2-49
- A Sample JSON-based REST Request, page 2-54
- A Sample XML-based REST Response, page 2-56
- A Sample JSON-based REST Response, page 2-58
- Samples of XML-based Fault Responses, page 2-58

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</REQUEST_CURRENCY_CODE>
      <SUPPLIER_ID>601</SUPPLIER_ID>
      <SUPPLIER_SITE_ID>1415</SUPPLIER_SITE_ID>
      <REQUESTOR_ID>xxxxxxxx</REQUESTOR_ID>
      <ASSIGNEE_RESOURCE_ID>xxxxxxxx</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>xxxxxxxx</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>
      </P_SDR_CUST_TBL_ITEM>
    </P_SDR_CUST_TBL>
  </InputParameters>
</CREATE_SD_REQUEST_Input>

```

```

<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": "xxxxxxxxxx",
        "ASSIGNEE_RESOURCE_ID": "xxxxxxxxxx",
        "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",

```



```

<?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 the 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 the GET method:

```

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

```

Note: The message payload used here is an example. You should use the actual definition of the service in XSD and WADL.

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>
```

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.

After a PL/SQL API interface definition is exposed as a SOAP web service represented in WSDL, the deployed service can be orchestrated into a meaningful BPEL process within a SOA Composite application with service endpoints. At runtime, the SOA Composite in the WebLogic managed server where the `soa-infra` application is running can be exposed to customers and invoked through any of the web service clients or orchestration tools including Oracle JDeveloper, Apache Axis, .NET Web Service Client, Oracle BPEL Process Manager, and Oracle Enterprise Service Bus (ESB).

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 runtime tasks are discussed in this chapter. For the example described in the following sections, Oracle JDeveloper 11g (11.1.1.6.0) is used as a design-time tool to create a SOA composite application with BPEL process and Oracle SOA Suite 11g (11.1.1.6.0) is used for the process deployment.

Note: While using Oracle JDeveloper with other Oracle Fusion Middleware components (such as Oracle SOA Suite), to enable SOA technologies, you need to manually download Oracle SOA Suite Composite Editor, an Oracle JDeveloper extension for SOA technologies. For more information on installing additional Oracle Fusion Middleware design time components, see the *Oracle Fusion Middleware Installation Guide for Oracle JDeveloper*.

This chapter includes the following topics:

- Using PL/SQL SOAP Services, page 3-2

- Invoking a Synchronous SOAP Web Service from a SOA Composite Application with BPEL Process, page 3-6
- Invoking an Asynchronous SOAP Web Service from a SOA Composite Application with BPEL Process, page 3-46
- Using PL/SQL REST Services, page 3-68
 - Invoking a REST Service Using HTTP Basic Authentication and XML Payload With REST Header, page 3-69
 - Invoking a REST Service Using Token Based Authentication and JSON Payload, page 3-79

Using PL/SQL SOAP Services

SOA Composite Application with BPEL Process Scenario

Take PL/SQL Supplier Ship and Debit Request API OZF_SD_REQUEST_PUB as an example to explain the creation of a SOA Composite application with BPEL process.

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.

- *Invoking a Synchronous SOAP Web Service*

A synchronous service is used to process the request if all needed information is supplied immediately. The request number will be returned to the requestor synchronously.

When the SOA Composite application with BPEL process has been successfully processed after deployment, a ship and debit request is created in the Oracle Order Management. The request number should be the same as the payload input value.

See: Invoking a Synchronous SOAP Web Service from a SOA Composite Application with BPEL Process, page 3-6.

- *Invoking an Asynchronous SOAP Web Service*

If a response message does not return to the requestor right away or if it takes longer time to process the request, an asynchronous web service can be used instead. Once the request is completed, the response information will be received.

A SOA Composite application with BPEL process will be created. The BPEL process whose input will be the response of an asynchronous operation. This BPEL process writes the response received to an output file on the server where the SOA Composite is deployed.

See: Invoking an Asynchronous SOAP Web Service from a SOA Composite

Prerequisites to Create a SOA Composite Application with 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 must create a security grant for a specific user (such as "TRADEMGR") or user group if necessary to ensure that the user has the access privilege to the interface.

- An integration administrator or integration developer needs to generate a web service first. The administrator will deploy the generated service to an Oracle SOA Suite WebLogic managed server.
- An integration developer needs to locate and record the deployed WSDL URL for the PL/SQL interface exposed as a web service.
- SOAHeader variables need to be populated for web service authorization.

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 the parameters within these record types for a successful invocation.

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 a PL/SQL Web Service Composite

An integration administrator or integration developer must first create a web service for a selected interface definition, and then the administrator can deploy the service from Oracle Integration Repository to an Oracle SOA Suite WebLogic managed server.

For example, the following steps must be performed first before the integration developer creates a BPEL process by using the deployed WSDL:

1. To generate a web service, the integration administrator or the integration developer locates the interface definition first (such as a PL/SQL interface `OZF_SD_REQUEST_PUB`) and selects desired interaction pattern information (either synchronous or asynchronous or both patterns) from the Interaction Pattern table.

This can be selected at the interface level or at the method level before clicking **Generate** in the interface details page.

Once the service has been successfully generated, the SOAP Service Status field changed from 'Not Generated' to 'Generated' in the SOAP Web Service tab. For detailed instructions 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, the administrator selects one authentication type before clicking **Deploy**. The deployed service in Oracle SOA Suite is an active service and is ready to accept new SOAP requests.

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

For detailed instructions 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 a WSDL URL

Apart from the required tasks mentioned above, the integration developer needs to locate and record the deployed service WSDL URL for the interface that needs to be orchestrated into a meaningful BPEL process in Oracle JDeveloper.

This can be done by clicking the **View WSDL** link in the interface details page. Copy the WSDL URL from the new pop-up window. This URL will be used later in creating a partner link service in a BPEL process.

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

Integration Repository > PLSQL Interface : OZF_SD_REQUEST Public API > PLSQL Method Details : Create SDR >

PLSQL Interface : OZF_SD_REQUEST Public API [Browse](#) [Search](#) [Printable Page](#)

Internal Name: OZF_SD_REQUEST_PUB Scope: Public
 Type: PL/SQL Interface Source: Oracle
 Product: Trade Management
 Status: Active
 Business Entity: [Supplier Ship and Debit Request](#)

Overview | **SOAP Web Service** | REST Web Service

SOAP Service Status: Deployed | Active | [View WSDL](#) Log Configuration: Disabled

Service Operations

Display Name	Internal Name	Synchronous	Asynchronous	Grant
▲ OZF_SD_REQUEST Public API	OZF_SD_REQUEST_PUB	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
Create SDR	CREATE_SD_REQUEST	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
Update SD Request	UPDATE_SD_REQUEST	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
copy_sd_request	COPY_SD_REQUEST	<input checked="" type="checkbox"/>	<input type="checkbox"/>	

TIP To apply any changes in Interaction Pattern, Generate or Regenerate the service.

Web Service Security

* Authentication Type

- Username Token
- SAML Token (Sender Vouches)

Copyright (c) 1998, 2019, Oracle and/or its affiliates. All rights reserved. [Privacy Statement](#)

For information on how to search for an interface and view the interface details, see Searching and Viewing Integration Interfaces, page 2-1.

Setting Variables in SOAHeader for a SOAP Request

Certain variables required to set application context must be populated for the SOAHeader elements to pass values during the service invocation. These SOAHeader elements for PL/SQL interface type are *Responsibility*, *RespApplication*, *SecurityGroup*, *NLSLanguage*, and *Org_Id*.

Note: The user name and password information is defined by the web service security policy (such as `oracle/wss_username_token_service_policy`). For detailed instructions on how to pass the security headers along with the SOAP request, see Configuring Web Service Policies, page 3-34.

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 "OZF_USER")
RespApplication	Application Short Name (such as "OZF")
SecurityGroup	Security Group Key (such as "STANDARD")
NLSLanguage	NLS Language (such as "AMERICAN")
Org_Id	Org Id (such as "204")

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 invocation 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 process.

Detailed information on how to set SOAHeader for the SOAP request, see Assigning SOAHeader Parameters, page 3-23.

Invoking a Synchronous Web Service from a SOA Composite Application with BPEL Process

Based on the single invoice creation scenario, a synchronous web service is used to process the request when all needed information is supplied immediately. The request number will be returned to the requestor synchronously.

The following design-time tasks are discussed in this chapter:

1. Create a SOA Composite Application with Synchronous BPEL Process, page 3-7

Use this step to create a new SOA Composite application with BPEL project called `ShipDebitRequest.bpel` using a 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 to the request application.

2. Create a Partner Link, page 3-10

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 a web service.

3. Add a Partner Link for File Adapter, page 3-11

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

4. Add Invoke activities, page 3-19

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

- Point to the File Adapter to synchronously 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 3-23

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 how to create SOA composite applications using BPEL process service component, see the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite* for details.

Creating a SOA Composite Application with BPEL Process

Use this step to create a new SOA Composite application that will contain various BPEL process activities.

To create a new SOA Composite application with BPEL project:

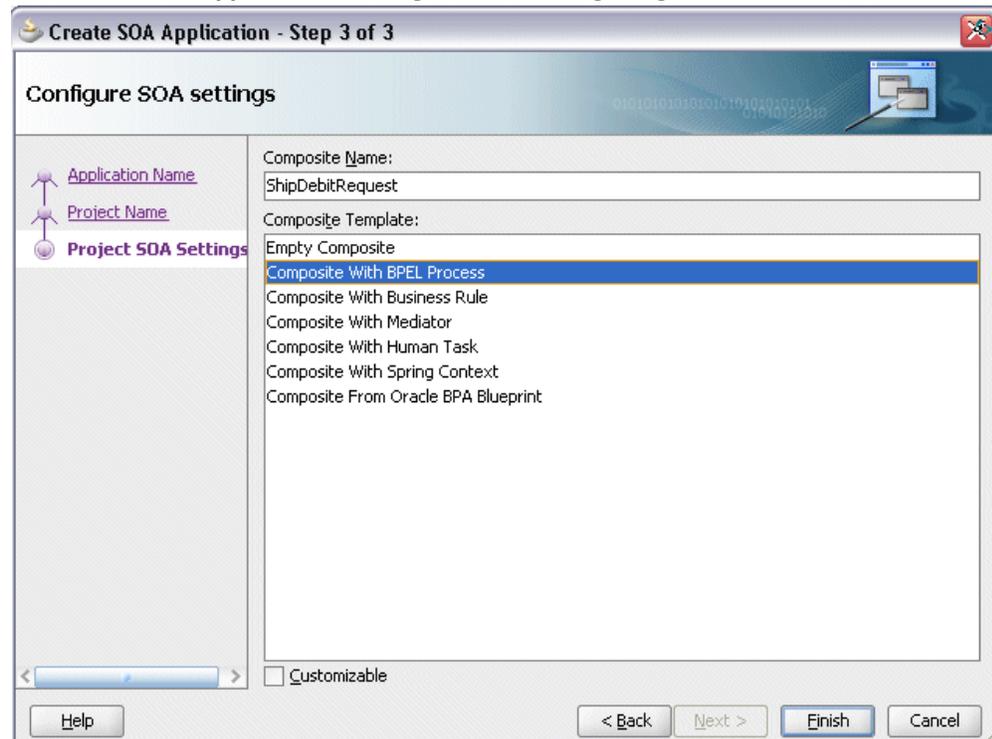
1. Open Oracle JDeveloper.
2. Click **New Application** in the Application Navigator.
The "Create SOA Application - Name your application" page is displayed.
3. Enter an appropriate name for the application in the Application Name field and select **SOA Application** from the Application Template list.

Click **Next**. The "Create SOA Application - Name your project" page is displayed.

4. Enter an appropriate name for the project in the Project Name field, for example, ShipDebitRequest.
5. In the Project Technologies tab, select 'Web Services' and ensure that **SOA** is selected from the Available technology list to the Selected technology list.

Click **Next**. The "Create SOA Application - Configure SOA settings" page is displayed.

The Create SOA Application - Configure SOA settings Page



6. Select **Composite With BPEL Process** from the Composite Template list, and then click **Finish**. You have created a new application, and a SOA project. This automatically creates a SOA composite.

The Create BPEL Process page is displayed.

The Create BPEL Process Page

BPEL Process

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

BPEL 1.1 Specification BPEL 2.0 Specification

Name: ShipDebitRequest

Namespace: http://xmlns.oracle.com/PLSQL/ShipDebitRequest/ShipDebitRequest

Template: Synchronous BPEL Process

Service Name: shipdebitrequest_client

Expose as a SOAP service

Transaction: required

Input: {http://xmlns.oracle.com/PLSQL/ShipDebitRequest/ShipDebitRequest}process

Output: http://xmlns.oracle.com/PLSQL/ShipDebitRequest/ShipDebitRequest}processResponse

Help OK Cancel

7. Leave the default **BPEL 1.1 Specification** selection unchanged. This creates a BPEL project that supports the BPEL 1.1 specification.

Enter an appropriate name for the BPEL process in the Name field, for example, ShipDebitRequest.

Select **Synchronous BPEL Process** in the Template field.

Select 'required' from the Transaction drop-down list. Click **OK**.

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

Note: Service 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. Navigate to SOA Content > Business Rules and click the composite.xml to view

the composite diagram.

Double click on the `ShipDebitRequest` component to open the BPEL process.

Creating a Partner Link for the Web Service

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

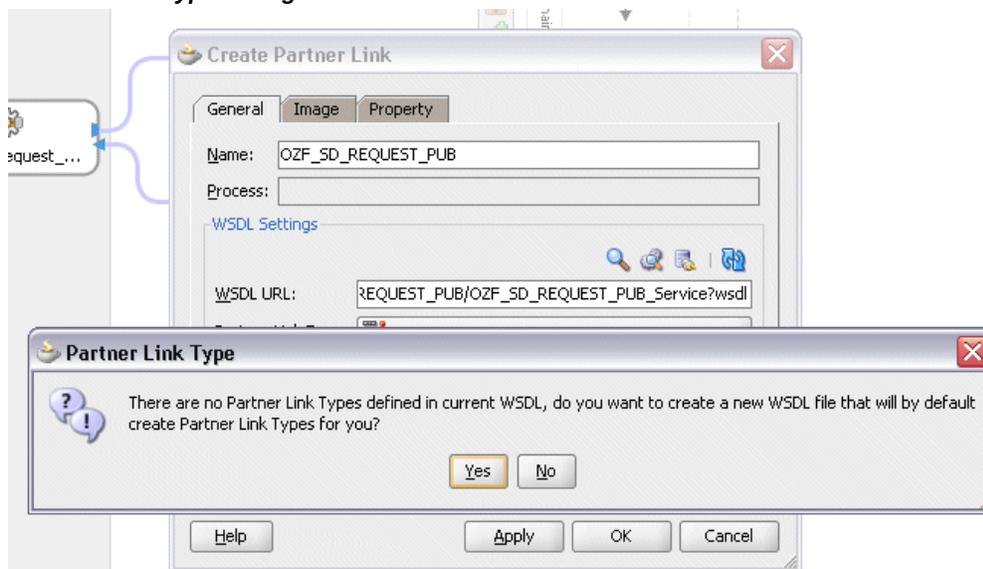
To create a partner link for `OZF_SD_REQUEST_PUB` web service:

1. In Oracle JDeveloper, place your mouse in the Partner Links area and right click to select **Create Partner Link...** from the pull-down menu. Alternatively, you can drag and drop **Partner Link** from the **BPEL Constructs** list into the right Partner Link swim lane of the process diagram.

The Create Partner Link window 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. Press the **[Tab]** key.
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.

Partner Link Type Dialog



Click **Yes** to have the Partner Name value populated automatically. Enter the partner link name as `OZF_SD_REQUEST_PUB`.

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

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

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

Adding a Partner Link for File Adapter

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

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

1. In Oracle JDeveloper, drag and drop the **File Adapter** service from the **BPEL Services** list into the right Partner Link swim lane 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`.
4. Click **Next**. The Adapter Interface dialog box appears.
5. Select the **Define from operation and schema (specified later)** radio button and click **Next**. The Operation dialog box appears.

Operation Dialog

The screenshot shows a dialog box titled "Adapter Configuration Wizard - Step 4 of 8". The main heading is "Operation". Below the heading, there is a paragraph of text: "The File Adapter supports four operations. There is a Read File operation that polls for incoming files in your local file system, a Write File operation that creates outgoing files, a Synchronous Read File operation that reads the current contents of a file, and a List Files operation that lists file names in specified locations. Specify the Operation type and Operation Name. Only one operation per Adapter Service may be defined using this wizard." Below this text, there are four radio button options under the label "Operation Type": "Read File", "Write File", "Synchronous Read File" (which is selected), and "List Files". Below the radio buttons, there is a text input field labeled "Operation Name:" with the text "SynchRead" entered. At the bottom of the dialog, there are five buttons: "Help", "< Back", "Next >", "Finish", and "Cancel".

6. 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

The screenshot shows a dialog box titled "File Directories" within the "Adapter Configuration Wizard - Step 5 of 8". The dialog contains the following elements:

- Header: "File Directories" with a decorative background of binary code and a gear icon.
- Instruction: "Enter directory information for the incoming file of the Synchronous Read File operation."
- Radio buttons: "Directory names are specified as: Physical Path Logical Name".
- Text field: "Directory for Incoming Files (physical path):" with the value "/usr/tmp" and a "Browse" button.
- Checkbox: "Archive processed files" (unchecked).
- Text field: "Archive Directory for Processed Files (physical path):" with an empty field and a "Browse" button.
- Checkbox: "Delete files after successful retrieval" (checked).
- Footer: "Help", "< Back", "Next >", "Finish", and "Cancel" buttons.

7. Select the **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 Oracle SOA Suite server.

Click **Next** to open the File Name dialog box.

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

File Name Dialog

Adapter Configuration Wizard - Step 6 of 8

File Name

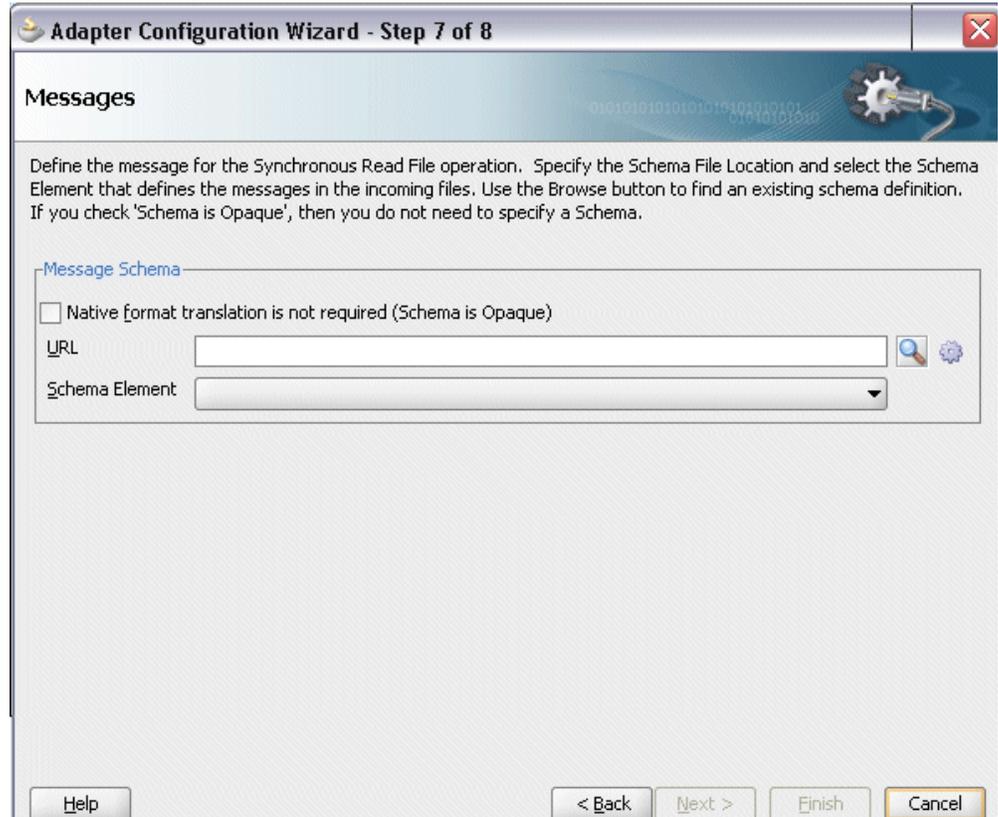
Enter the name of the file for the Synchronous Read File operation.

File Name:

Help < Back Next > Finish Cancel

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

Messages Dialog



9. Select **Browse for schema file** in front of the URL field.

The Type Chooser window is displayed.

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

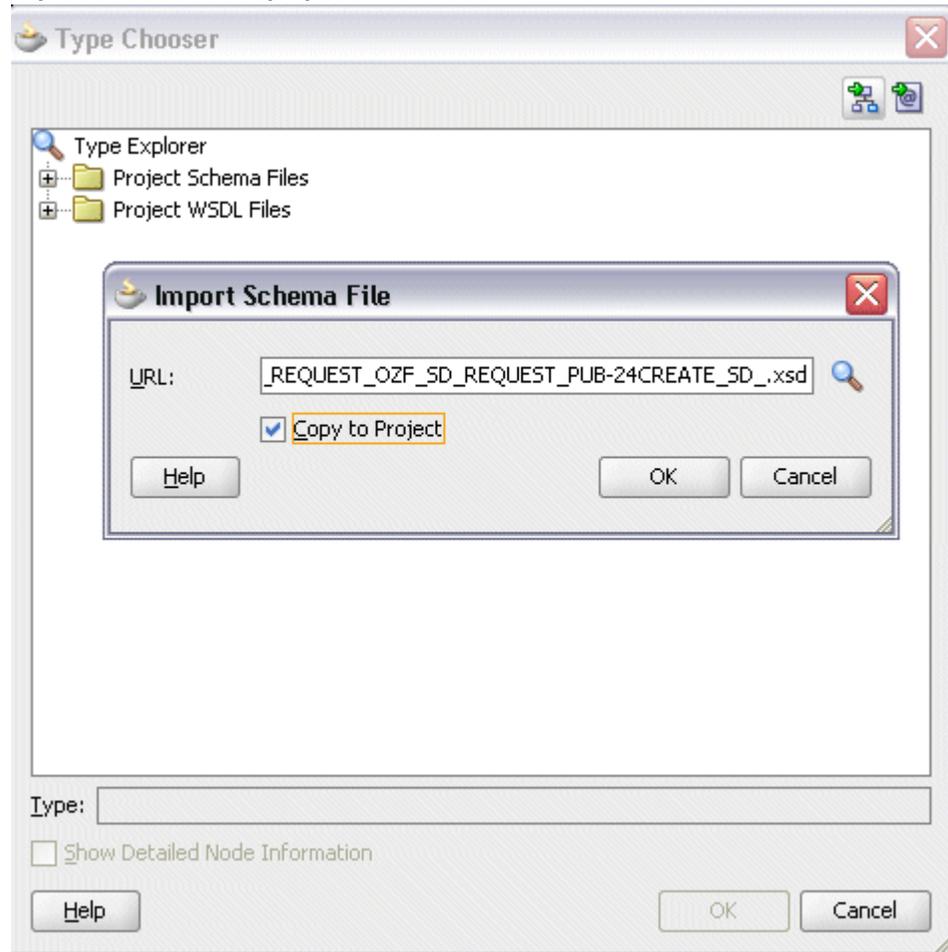
Enter the schema location for the service. Such as http :

```
//<soa_suite_hostname>:<port>/soa-  
infra/services/default/<jndi_name>_PLSQL_OZF_SD_REQUEST_PUB/OZ  
F_SD_REQUEST_PUB_Service?  
XSD=xsd/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://<soa_suite_hostname>:<port>/soa-infra/services/default/<jndi_name>_PLSQL_OZF_SD_REQUEST_PUB/OZF_SD_REQUEST_PUB_Service/?wsdl).

Select the **Copy to Project** checkbox and click **OK**.

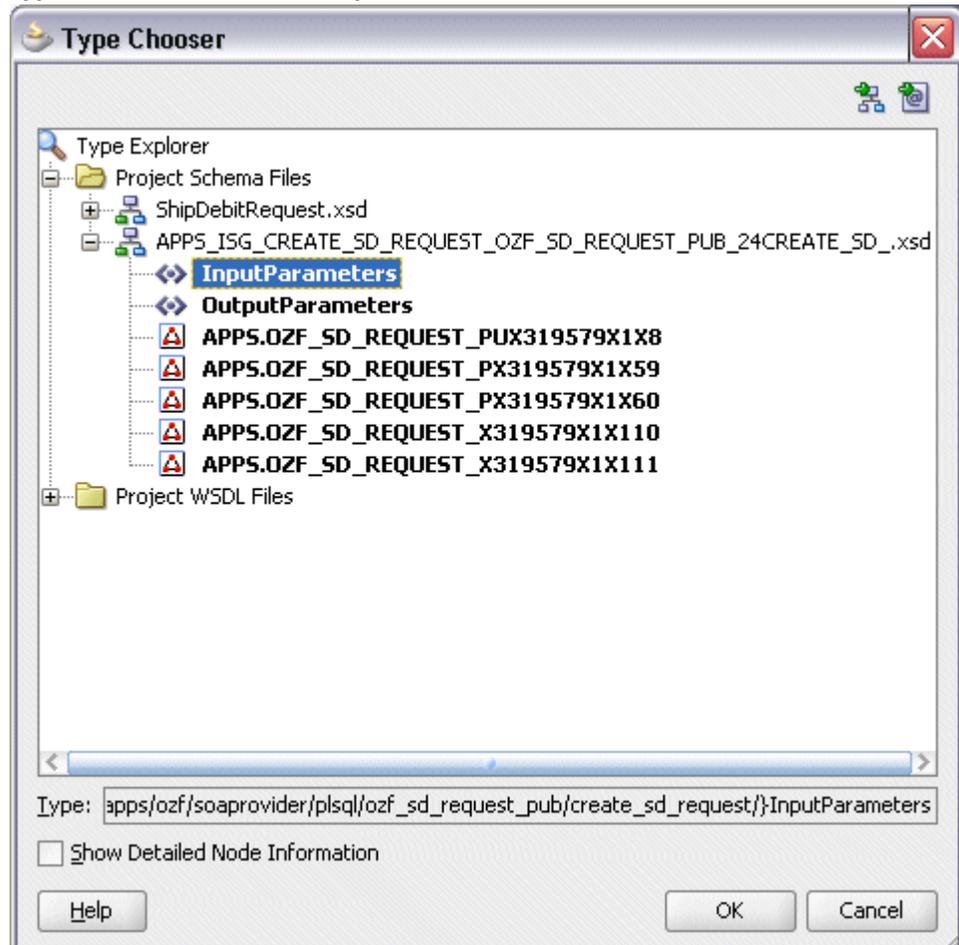
Import Schema File Pop-up Window



Select the **Maintain original directory structure for imported files** Copy Options checkbox and click **OK**.

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

Type Chooser Window with Imported Schema



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 URL, and the InputParameters is selected as Schema Element.

Messages Dialog with Selected Schema and Element

Adapter Configuration Wizard - Step 7 of 8

Messages

Define the message for the Synchronous Read File operation. Specify the Schema File Location and select the Schema Element that defines the messages in the incoming files. Use the Browse button to find an existing schema definition. If you check 'Schema is Opaque', then you do not need to specify a Schema.

Message Schema

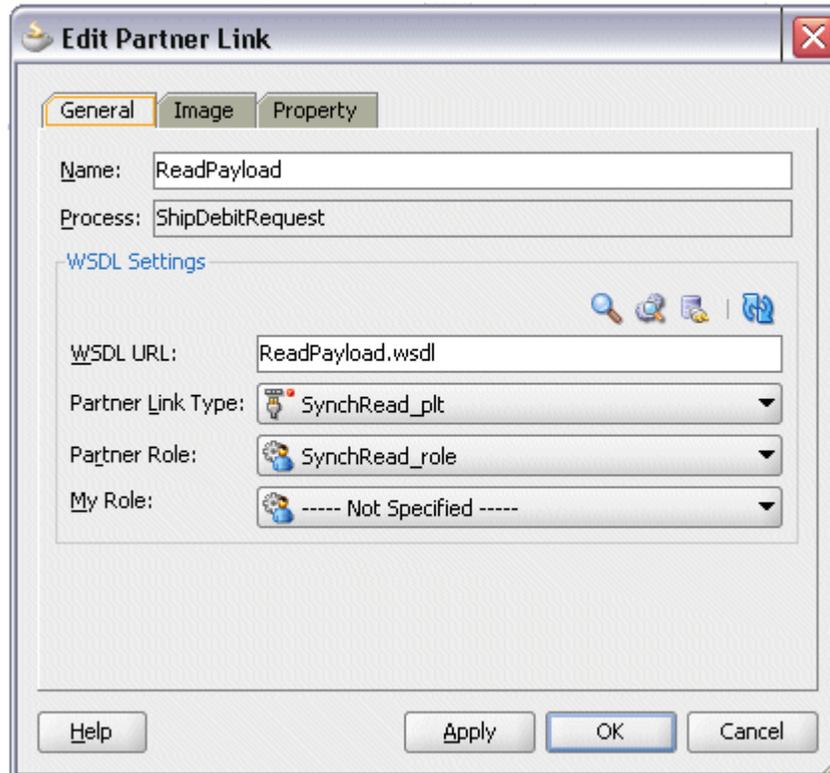
Native format translation is not required (Schema is Opaque)

URL  

Schema Element

10. 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`.

Edit 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.

11. Under applications window, navigate to file ReadPayload_file.jca. Set the value of property "DeleteFile" to "false".

ReadPayload_file.jca with Property Value

```
1 <adapter-config name="ReadPayload" adapter="File Adapter" wsdlLocation="ReadPayload.wsdl" xmlns="http://platform.integration.
2
3 <connection-factory location="eis/FileAdapter" adapterRef="" />
4 <endpoint-interaction portType="SynchRead_ptt" operation="SynchRead">
5 <interaction-spec className="oracle.tip.adapter.file.outbound.FileReadInteractionSpec">
6 <property name="PhysicalDirectory" value="/usr/tmp"/>
7 <property name="DeleteFile" value="false"/>
8 <property name="FileName" value="InputCreateSDRequest.xml"/>
9 </interaction-spec>
10 </endpoint-interaction>
11
12 </adapter-config>
```

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 Oracle JDeveloper, expand the **BPEL Constructs** from the Component Palette. Drag and drop the **Invoke** activity into the center swim lane of the process diagram, between the **receiveInput** and **replyOutput** 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



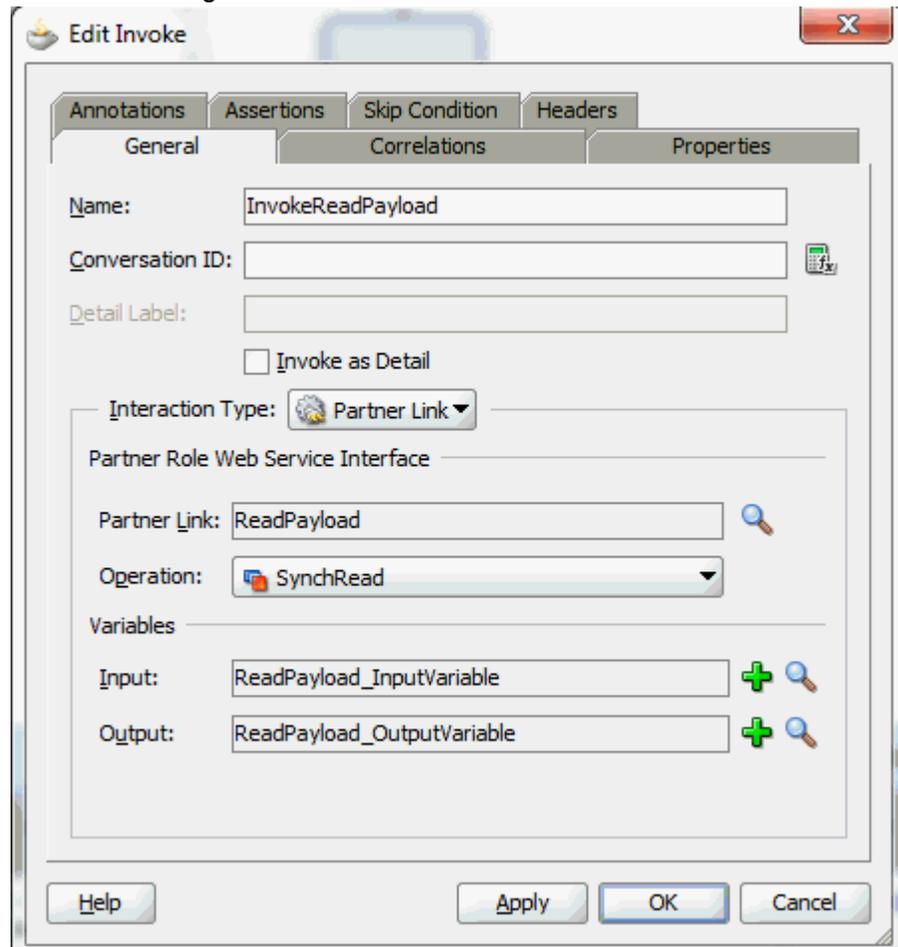
Select **Global Variable** and click **OK**.

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**.

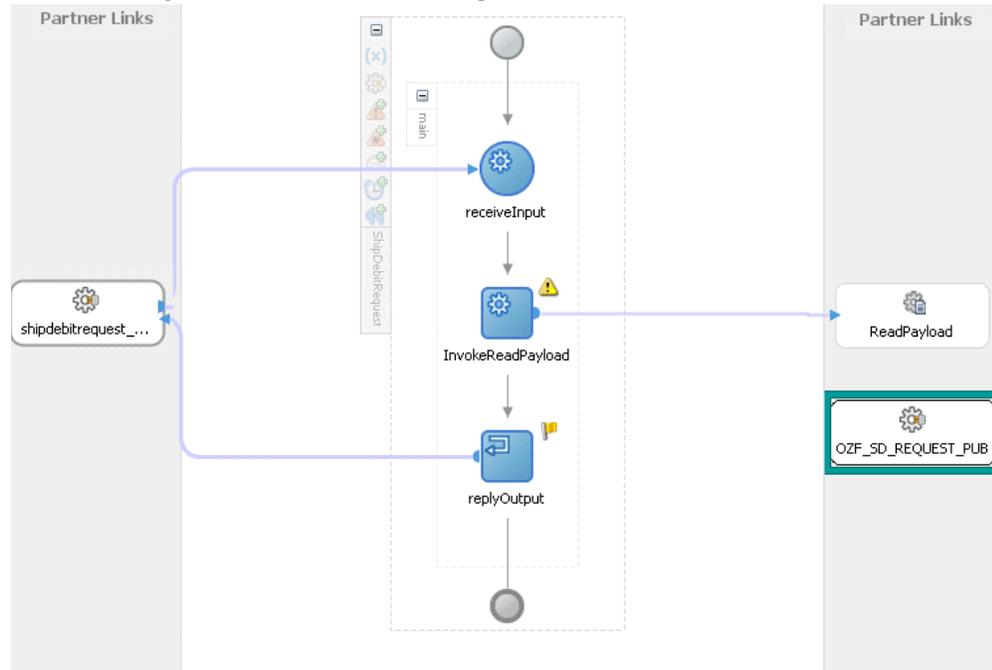
Edit Invoke Dialog with General Tab



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.

Invoke Activity Added to the Process Diagram



To add an Invoke activity for OZF_SD_REQUEST_PUB Partner Link:

1. In Oracle JDeveloper, expand the **BPEL Constructs** from the Component Palette. Drag and drop the **Invoke** activity into the center swim lane of the process diagram, after the first **Invoke** activity and the **replyOutput** 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.
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 Assign Activities

This step is to configure four **Assign** activities:

1. To set the SOAHeader details for ship and debit 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 application context during service invocation. These SOAHeader elements are *Responsibility*, *RespApplication*, *SecurityGroup*, *NLSLanguage*, and *Org_Id*.

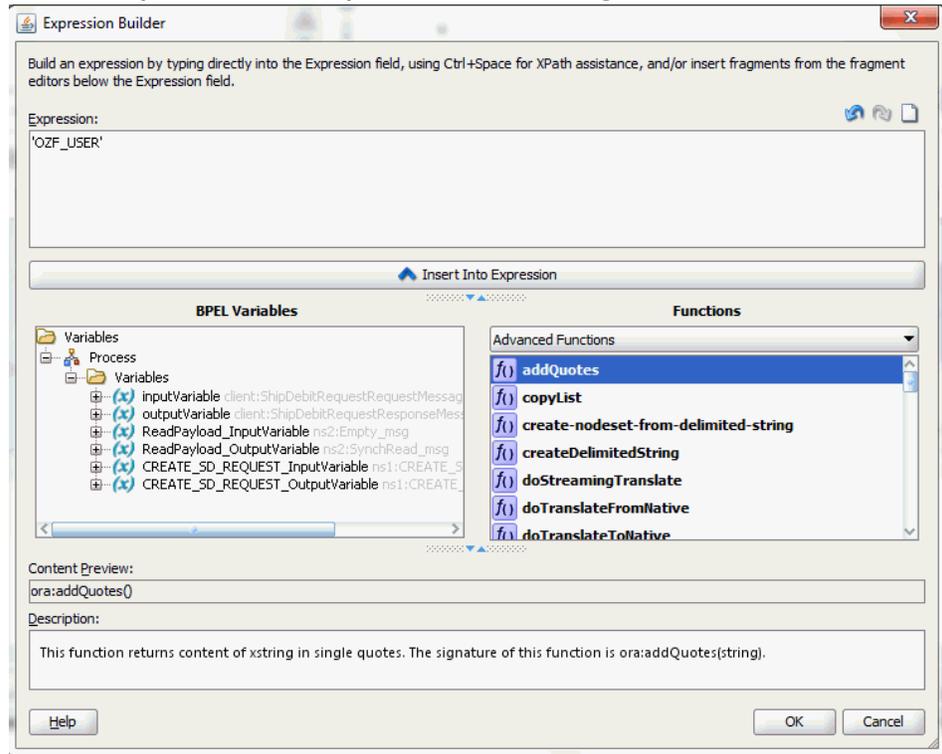
2. To set input payload for SOAP request.
3. To set input for SOAP request.
4. To set the SOAP response to output.

To add the first Assign activity to set SOAHeader details:

Assigning SOAHeader Parameters:

1. In Oracle JDeveloper, expand the **BPEL Constructs** from the Component Palette. Drag and drop the **Assign** activity into the center swim lane of the process diagram between the two **Invoke** activities you just created earlier.
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. Select the Copy Rules tab to expand the target trees:
 - Click the Expression icon to invoke the Expression Builder dialog.

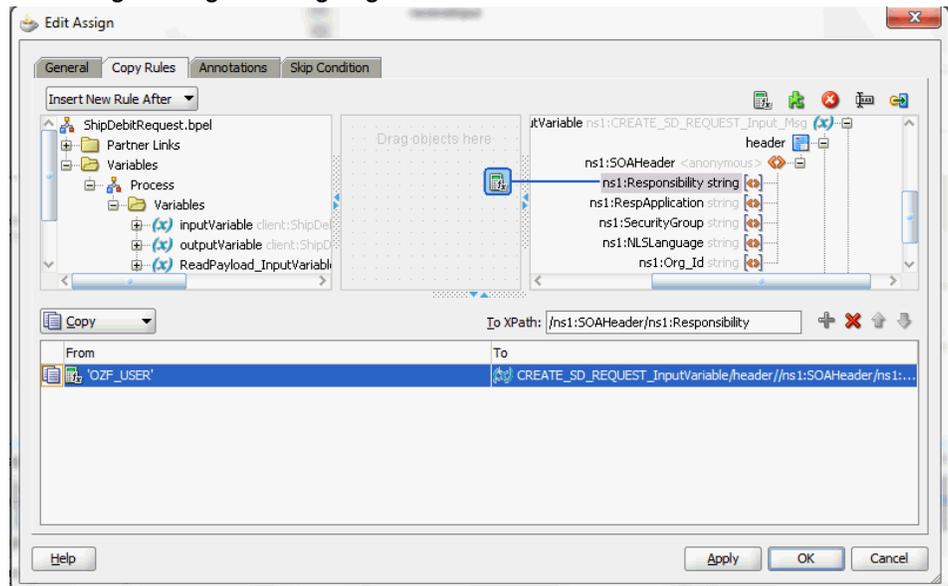
Creation Expression in the Expression Builder Dialog



Enter 'OZF_USER' in the Expression box. Click OK. The Expression icon with the expression value ('OZF_USER') appears in the center of the Edit Assign dialog, between the From and To navigation tree nodes.

- In the To navigation tree, navigate to **Variables > Process > Variables > CREATE_SD_REQUEST_InputVariable > header > ns1:SOAHeader** and select **ns1:Responsibility**. The To XPath value is displayed.
- Drag the Expression icon to connect to the target node (ns1:Responsibility) that you just identified. This creates a line that connects the source and target nodes. The copy rule is displayed in the From and To sections at the bottom of the Edit Assign dialog box.

Edit Assign Dialog for Assigning Parameters



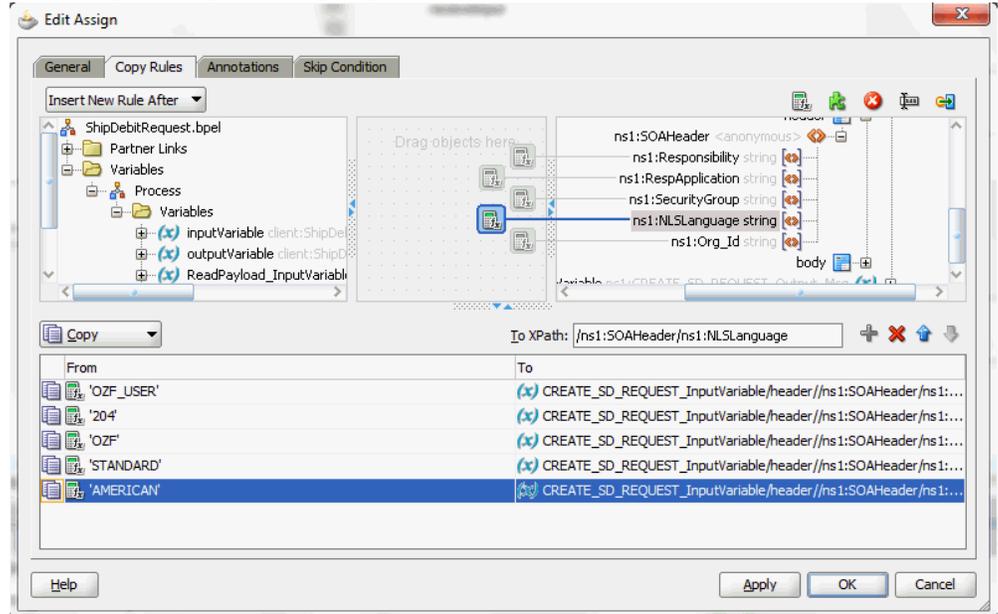
5. Enter second pair of parameters by clicking the Expression icon to invoke the Expression Builder dialog.
 - Enter 'OZF' in the Expression box. Click **OK**. The Expression icon with the expression value ('OZF') appears in the center of the Edit Assign dialog, between the From and To navigation tree nodes.
 - In the To navigation tree, navigate to **Variables > Process > Variables > CREATE_SD_REQUEST_InputVariable > header > ns1:SOAHeader** and select **ns1:RespApplication**. The To XPath field should contain your selected entry.
 - Drag the Expression icon to connect to the target node (ns1:RespApplication) that you just identified. This creates a line that connects the source and target nodes. The copy rule is displayed in the From and To sections at the bottom of the Edit Assign dialog box.

6. Enter the third pair of parameters by clicking the Expression icon to invoke the Expression Builder dialog.
 - Enter 'STANDARD' in the Expression box. Click **OK**. The Expression icon with the expression value ('STANDARD') appears in the center of the Edit Assign dialog, between the From and To navigation tree nodes.
 - In the To navigation tree, navigate to **Variables > Process > Variables > CREATE_SD_REQUEST_InputVariable > header > ns1:SOAHeader** and

select **ns1:SecurityGroup**. The XPath field should contain your selected entry.

- Drag the Expression icon to connect to the target node (ns1:SecurityGroup) that you just identified. This creates a line that connects the source and target nodes. The copy rule is displayed in the From and To sections at the bottom of the Edit Assign dialog box.
7. Enter the fourth pair of parameters by clicking the Expression icon to invoke the Expression Builder dialog.
- Enter 'AMERICAN' in the Expression box. Click **OK**. The Expression icon with the expression value ('AMERICAN') appears in the center of the Edit Assign dialog, between the From and To navigation tree nodes.
 - In the To navigation tree, navigate to **Variables > Process > Variables > CREATE_SD_REQUEST_InputVariable > header > ns1:SOAHeader** and select **ns1:NLSLanguage**. The XPath field should contain your selected entry.
 - Drag the Expression icon to connect to the target node (ns1:NLSLanguage) that you just identified. This creates a line that connects the source and target nodes. The copy rule is displayed in the From and To sections at the bottom of the Edit Assign dialog box.
8. Enter the fifth pair of parameters by clicking the Expression icon to invoke the Expression Builder dialog.
- Enter '204' in the Expression box. Click **OK**. The Expression icon with the expression value ('204') appears in the center of the Edit Assign dialog, between the From and To navigation tree nodes.
 - 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.
 - Drag the Expression icon to connect to the target node (ns1:Org_Id) that you just identified. This creates a line that connects the source and target nodes. The copy rule is displayed in the From and To sections at the bottom of the Edit Assign dialog box.
9. The Edit Assign dialog box appears.

Edit Assign Dialog for Assigning Parameters



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

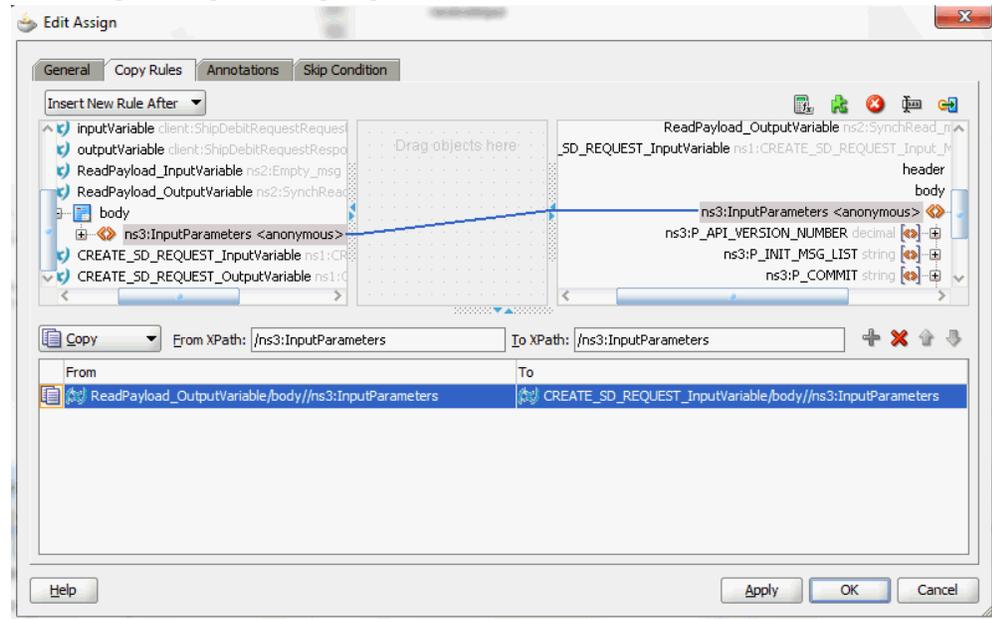
To enter the second Assign activity to pass payload information to the Invoke_EBS_SDR_Service Invoke activity:

1. Add the second **Assign** activity by dragging and dropping the **Assign** activity from the **BPEL Constructs** in the Component Palette into the center swim lane of the process diagram, between the 'SetSOAHeader' **Assign** activity and the 'Invoke_EBS_SDR_Service' **Invoke** activity.
2. Repeat Step 2 to Step 3 described in creating the first **Assign** activity to add the second **Assign** activity called 'SetPayload'.
3. Select the Copy Rules tab and expand the source and target trees:
 - In the From navigation tree, navigate to **Variable > Process > Variables > InputVariable > ReadPayload_OutVariable > body** and select **ns3: InputParameters**. The From XPath field is also displayed.
 - In the To navigation tree, navigate to **Variable > Process > Variables > CREATE_SD_REQUEST_InputVariable > body** and select **ns3: InputParameters**. The To XPath field is also displayed.

Drag the source node (InputParameters) to connect to the target node (InputParameters) that you just identified. This creates a line that connects the source and target nodes. The copy rule is displayed in the From and To sections at

the bottom of the Edit Assign dialog box.

Edit Assign Dialog for Assigning Parameters



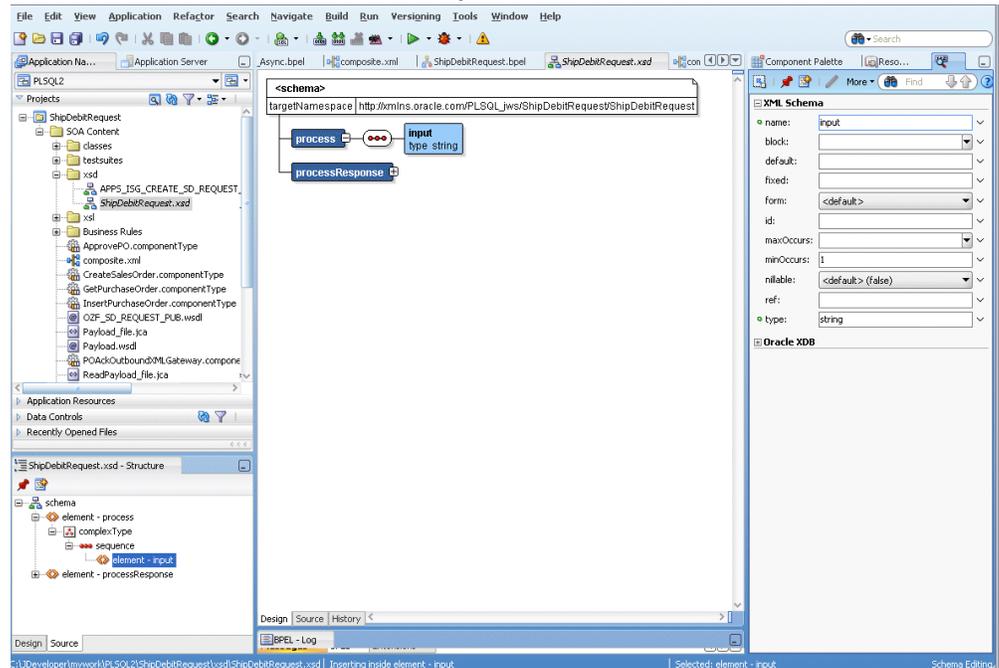
4. Click **Apply** and then **OK** to complete the configuration of the second **Assign** activity.

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 > SOA Content > xsd** folder to open the `ShipDebitRequest.xsd` file.
2. In the Design mode, expand 'process' to view elements within process request.

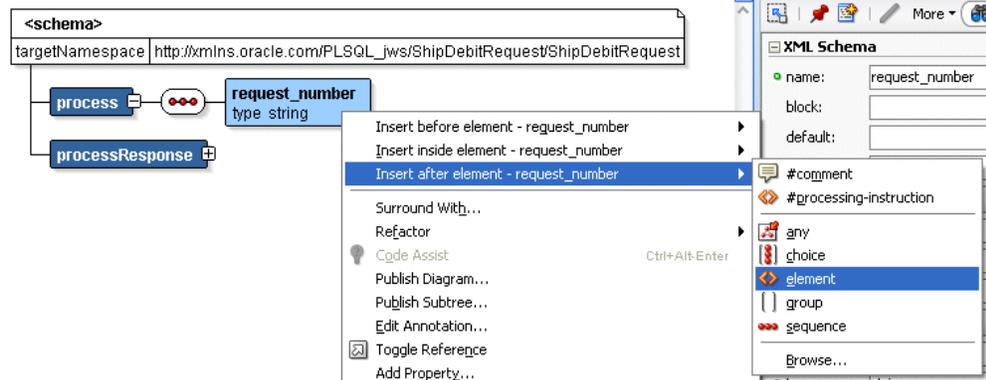
Schema Creation for BPEL Process Request



3. Click on element 'input' and change the property name from 'input' to 'request_number' in the XML Schema window.
4. Select and right-click on the 'request_number' element to open the pop-up menu. Select the **Insert after element – request_number > element** option. New element 'element1' is displayed in the schema design window underneath the 'request_number' element.

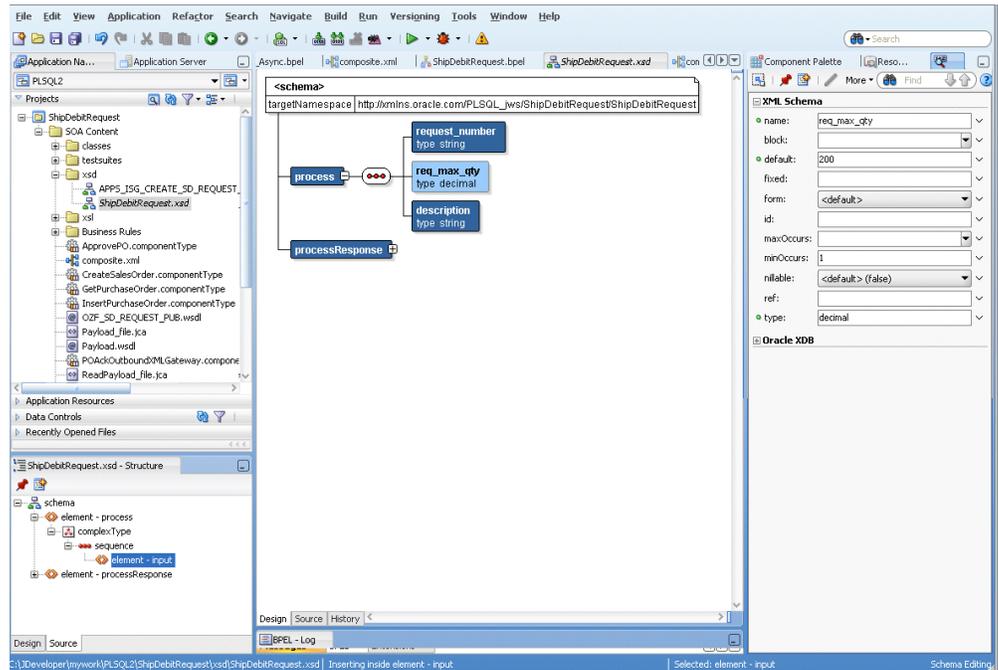
From the element properties section, change the name from 'element1' to 'description' and enter type as 'string'.

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 the **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 **BPEL Constructs** in the Component Palette into the center swim lane of the process diagram, between the second **Assign** activity 'SetPayload' and the Invoke_EBS_SDR_Service **Invoke** activity.
2. Repeat Step 2 to Step 3 described in creating the first **Assign** activity to add the third **Assign** activity called 'SetInput'.
3. Select the Copy Rules tab and expand the source and target trees:
 - In the From navigation tree, navigate to **Variable > Process > Variables > inputVariable > Payload > client:ShipDebitRequestProcessRequest** and select **client:request_number**. The From XPath field is also displayed.
 - In the To navigation tree, navigate to **Variable > Process > Variables > Create_SD_REQUEST_InputVariable > Body > ns4:InputParameters > ns4:P_SDR_HDR_REC** and select **ns4:REQUEST_NUMBER**. The To XPath field is also displayed.

Drag the source node (client:request_number) to connect to the target node (ns4:

REQUEST_NUMBER) that you just identified. This creates a line that connects the source and target nodes. The copy rule is displayed in the From and To sections at the bottom of the Edit Assign dialog box.

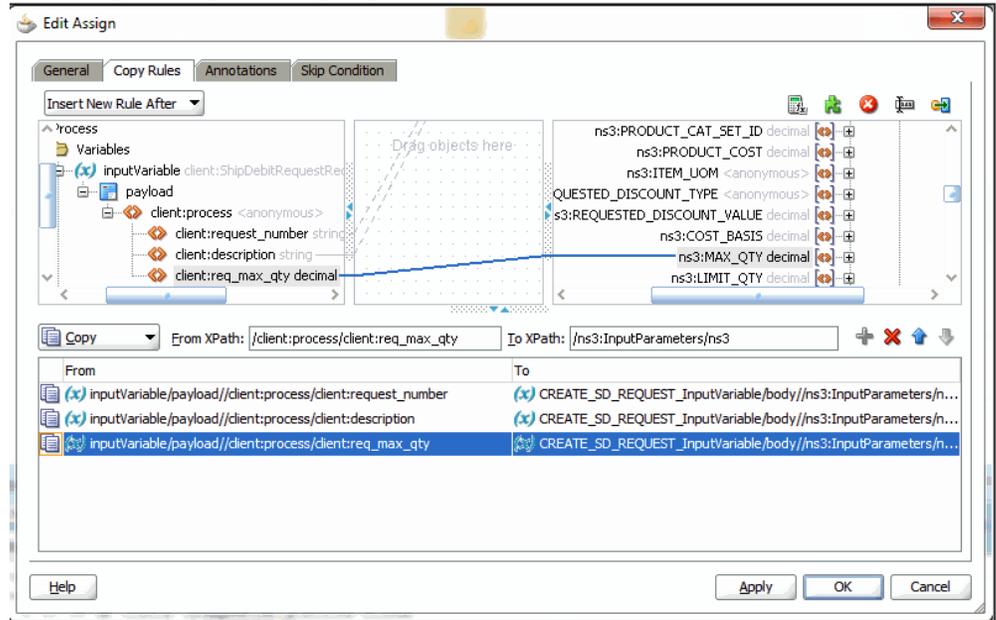
4. Enter the second pair of parameters with the following values:
 - In the From navigation tree, navigate to **Variable > Process > Variables > inputVariable > Payload > client:ShipDebitRequestProcessReqst** and select **client:description**. The XPath field should contain your selected entry.
 - In the To navigation tree, navigate to **Variable > Process > Variables > Create_SD_REQUEST_InputVariable > Body > ns4:InputParameters > ns4:P_SDR_HDR_REC** and select **ns4:REQUEST_DESCRIPTION**. The XPath field should contain your selected entry.

Drag the source node (client:description) to connect to the target node (ns4:REQUEST_DESCRIPTION) that you just identified. This creates a line that connects the source and target nodes. The copy rule is displayed in the From and To sections at the bottom of the Edit Assign dialog box.

5. Enter the third pair of parameters with the following values:
 - In the From navigation tree, navigate to **Variable > Process > Variables > inputVariable > Payload > client:ShipDebitRequestProcessReqst** and select **client:req_max_qty**. The XPath field should contain your selected entry.
 - In the To navigation tree, navigate to **Variable > Process > Variables > Create_SD_REQUEST_InputVariable > Body > ns4:InputParameters > ns4:P_SDR_LINES_TBL > ns4:P_SDR_LINES_TBL_ITEM** and select **ns4:MAX_QTY**. The XPath field should contain your selected entry.

Drag the source node (client:req_max_qty) to connect to the target node (ns4:MAX_QTY) that you just identified. This creates a line that connects the source and target nodes. The copy rule is displayed in the From and To sections at the bottom of the Edit Assign dialog box.

Edit Assign Dialog for Assigning Parameters



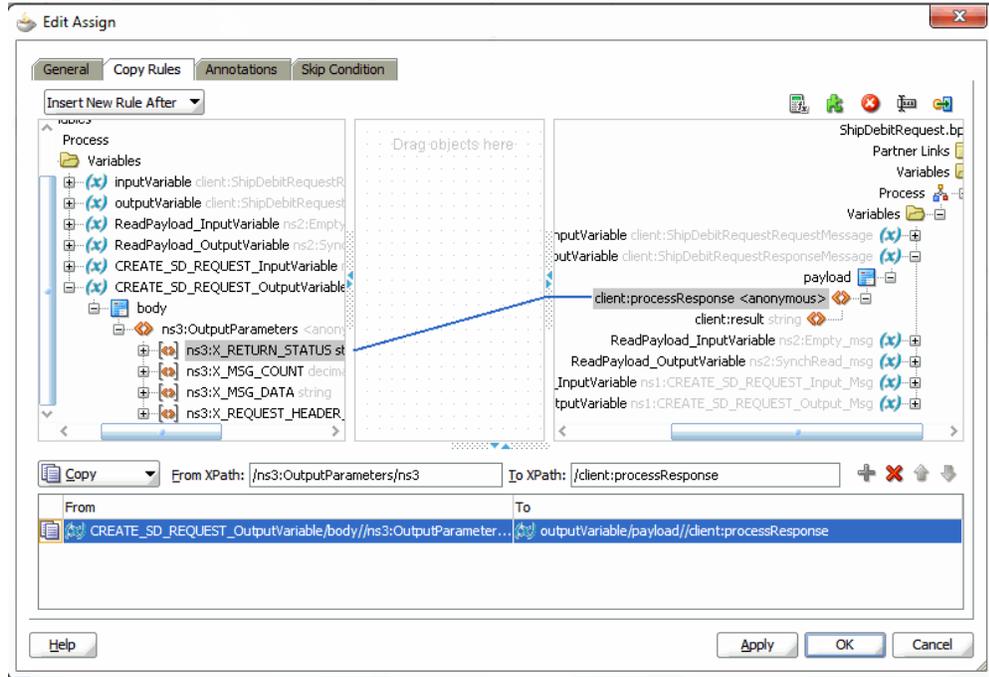
6. Click **Apply** and then **OK** to complete the configuration of the **Assign** activity.

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

1. Add the fourth **Assign** activity by dragging and dropping the **Assign** activity from the **BPEL Constructs** in the Component Palette into the center swim lane of the process diagram, between the **Invoke_EBS_SDR_Service Invoke** and the **replyOutput** activities.
2. Repeat Step 2 to Step 3 described in creating the first **Assign** activity to add the fourth **Assign** activity called 'SetResponse'.
3. Select the Copy Rules tab and expand the source and target trees:
 - In the From navigation tree, navigate to **Variable > Process > Variables > CREATE_SD_REQUEST_OutputVariable > body > ns3:OutputParameters** and select **ns3:X_RETURN_STATUS**.
 - In the To navigation tree, navigate to **Variable > Process > Variables > outputVariable > payload** and select **client:processResponse**.

Drag the source node (ns3:X_RETURN_STATUS) to connect to the target node (client:processResponse) that you just identified. This creates a line that connects the source and target nodes. The copy rule is displayed in the From and To sections at the bottom of the Edit Assign dialog box.

Edit Assign Dialog for Assigning Parameters



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

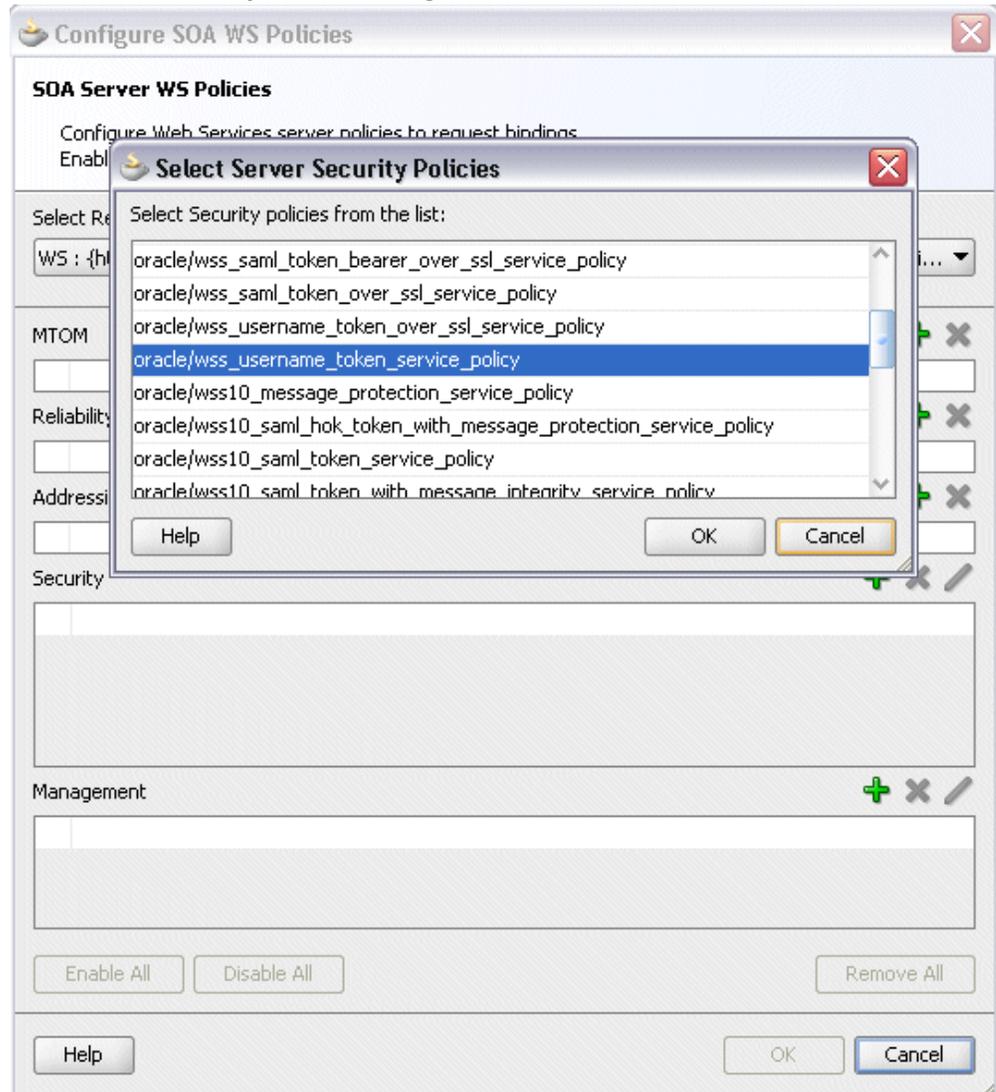
Configuring Web Service Policies

Use the following steps to add a security policy at design time:

1. Navigate to SOA Content > Business Rules > composite.xml. Right click on the OZF_SD_REQUEST_PUB service and select "Configure WS Policies" from the drop-down list.
2. The Configure SOA WS Policies dialog appears.

In the Security section, click the **Add** icon (+). The Select Server Security Policies dialog appears.

Select Server Security Policies Dialog

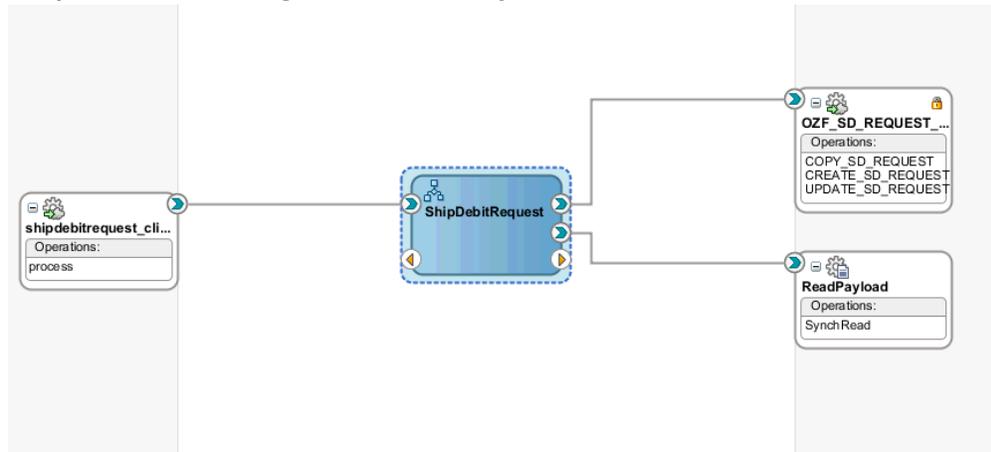


Select 'oracle/wss_username_token_service_policy' and click **OK**.

The attached security policy is shown in the Security section.

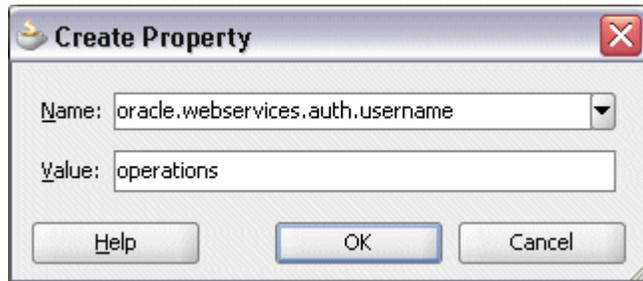
A lock icon appears in the OZF_SD_REQUEST_PUB service of composite.xml indicating that a security policy has been successfully attached.

composite.xml Flow Diagram with a Security Lock Icon



3. From the navigation menu, select **View > Property Inspector** to display the Property Inspector window for the OZF_SD_REQUEST_PUB service component. In the Properties section, click the **Add** icon (+) for binding properties. The Create Property dialog appears. Enter 'oracle.webservices.auth.username' in the Name field and enter 'operations' as the value.

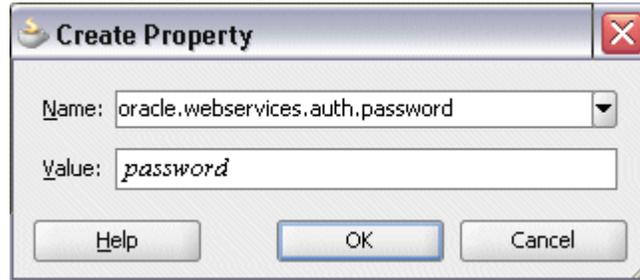
Create Property Dialog for oracle.webservices.auth.username



Click **OK**.

4. Use the same approach by clicking the **Add** icon (+) again in the Properties section for binding properties. Enter 'oracle.webservices.auth.password' in the Name field. Enter the associated password for user 'operations' in the Value field.

Create Property Dialog for oracle.webservices.auth.password



Click OK.

Both selected property names and values appear in the Properties section.

Click the Source tab of `composite.xml` and notice that the `oracle.webservices.auth.username` and `oracle.webservices.auth.password` property names and the associated values are added to the `OZF_SD_REQUEST_PUB` reference.

Property Information Shown in `composite.xml`

```
27 <reference name="OZF_SD_REQUEST_PUB"
28     ui:wSDLLocation="OZF_SD_REQUEST_PUB.wsdl">
29     <interface.wSDL interface="http://xmlns.oracle.com/apps/ozf/soapprovider/plsql/ozf_sd_request_pub/#wsdl.interfa
30     <binding.ws port="http://xmlns.oracle.com/apps/ozf/soapprovider/plsql/ozf_sd_request_pub/#wsdl.endpoint(OZF_SD_
31     location="OZF_SD_REQUEST_PUB.wsdl" soapVersion="1.1">
32     <wsp:PolicyReference URI="oracle/wss_username_token_client_policy"
33     orawsp:category="security" orawsp:status="enabled"/>
34     <property name="oracle.webservices.auth.username" type="xs:string"
35     many="false" override="may">operations</property>
36     <property name="oracle.webservices.auth.password" type="xs:string"
37     many="false" override="may">password</property>
38     </binding.ws>
39 </reference>
```

Deploying and Testing the SOA Composite with Synchronous BPEL Process

To invoke the synchronous Supplier Ship and Debit Request service (`OZF_SD_REQUEST_PUB`) from the BPEL client contained in the SOA composite, the SOA composite needs to be deployed to the Oracle WebLogic managed server. This can be achieved using Oracle JDeveloper. Once the composite is deployed, it can be tested from the Oracle Enterprise Manager Fusion Middleware Control Console.

Prerequisites

Before deploying the SOA composite with BPEL process using Oracle JDeveloper, you must have established the connectivity between the design-time environment and the runtime server. For information on how to configure the necessary server connection, see *Configuring Server Connection*, page B-1.

Note: If a local instance of the WebLogic Server is used, start the

WebLogic Server by selecting Run > Start Server Instance from Oracle JDeveloper. Once the WebLogic Admin Server "DefaultServer" instance is successfully started, the <Server started in Running mode> and DefaultServer started message in the Running:DefaultServer and Messages logs should appear.

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

Perform the following runtime tasks:

1. Deploy the SOA Composite Application with BPEL Process, page 3-38
2. Test the SOA Composite Application with BPEL Process, page 3-42

Deploying the SOA Composite with BPEL Process

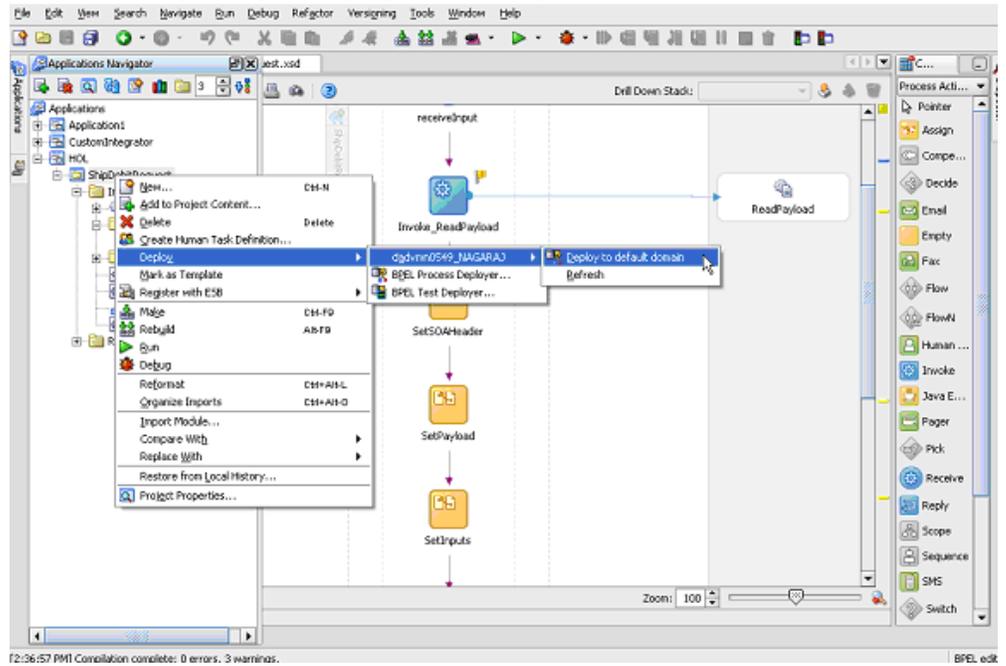
You must deploy the SOA composite with BPEL process (`ShipDebitRequest.bpel`) that you created earlier before you can run it.

To deploy the SOA composite application:

1. In the Applications Navigator of JDeveloper, select the **ShipDebitRequest** project.
2. Right-click the project and select **Deploy > [project name] > [serverConnection]** from the menu.

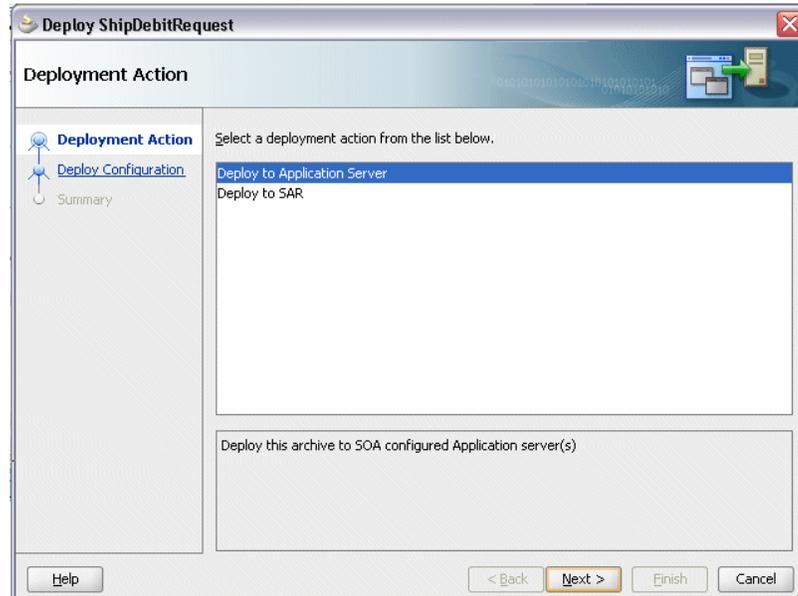
For example, you can select **Deploy > ShipDebitRequest > SOAServer** to deploy the process if you have the connection set up appropriately.

Oracle JDeveloper Page to Deploy a SOA Composite with BPEL Process



Note: If this is the first time to set up the server connection, then the Deployment Action dialog appears. Select 'Deploy to Application Server' and click Next.

Deployment Action Dialog



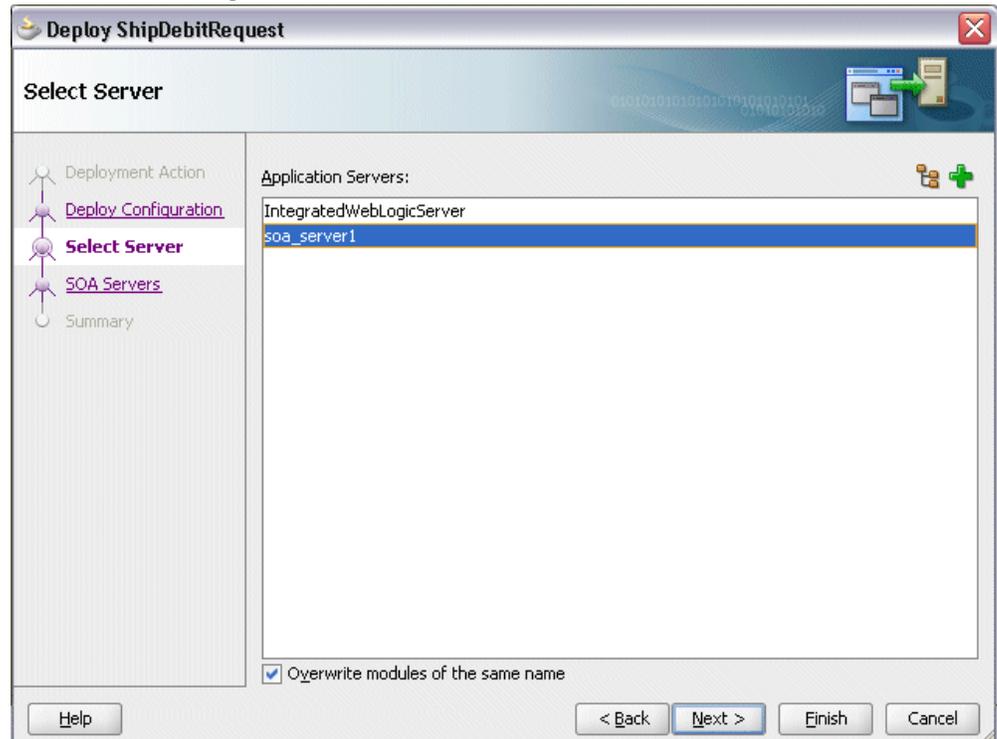
In the Deploy Configuration dialog, ensure the following information is selected before clicking **Next** to add a new application server:

- New Revision ID: 1.0
- Mark composite revision as default: Select this checkbox.
- Overwrite any existing composites with the same revision ID: Select this checkbox.

The steps to create a new Oracle WebLogic Server connection from JDeveloper are covered in Configuring Server Connection, page B-1.

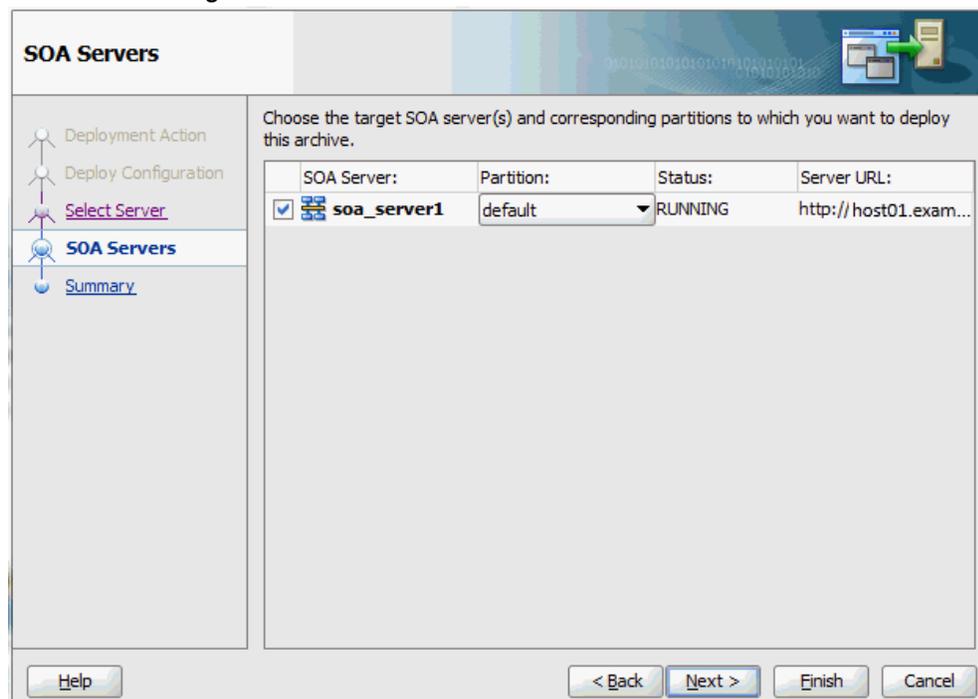
3. In the Select Server dialog, select 'soa-server1' that you have established the server connection earlier. Click **Next**.

Select Server Dialog



4. In the SOA Servers dialog, accept the default target SOA Server ('soa-server1') selection.

SOA Server Dialog



Click **Next** and **Finish**.

5. If you are deploying the composite for the first time from your Oracle JDeveloper session, the Authorization Request window appears. Enter the user name and password information specified during the Oracle SOA Suite installation. Click **OK**.
6. Deployment processing starts. Monitor deployment process and check for successful compilation in the SOA - Log window.

Verify that the deployment is successful in the Deployment - Log window.

Testing the SOA Composite Application

Once the BPEL process contained in the SOA composite application has been successfully deployed, you can manage and monitor the process from Oracle Enterprise Manager Fusion Middleware Control Console. You can also test the process and the integration interface by manually initiating the process.

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 information is also written in the XML file.

To test the SOA composite application:

1. Navigate to Oracle Enterprise Manager Fusion Middleware Control Console (

http://<servername>:<port>/em). The login page appears.

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

2. Enter the user name and password information specified during the installation. Click **Login** to log in to a farm. The composite (ShipDebitRequest) you deployed is displayed in the Applications Navigation tree.

You may need to select an appropriate target instance farm if there are multiple target Oracle Enterprise Manager Fusion Middleware Control Console farms.

3. From the Farm navigation pane, expand the SOA node, and then the soa-infra node in the tree to navigate through the SOA Infrastructure home page and menu to access your deployed SOA composite applications running on soa-infra managed server.

Click the ShipDebitRequest [1.0] link.

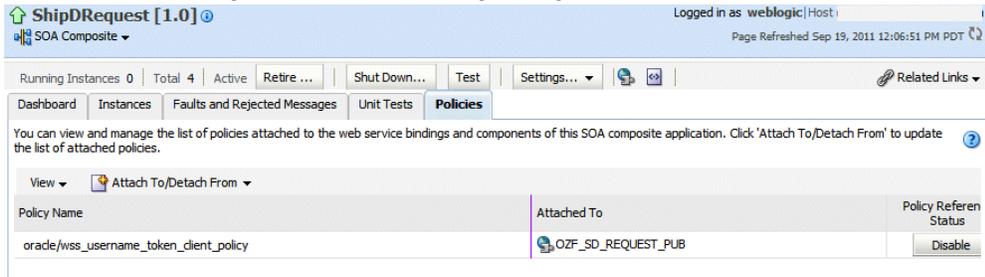
Oracle Enterprise Manager Fusion Middleware Control Console to View the Deployed Service

The screenshot displays the Oracle Enterprise Manager Fusion Middleware Control Console interface. The left-hand navigation pane shows a tree structure under 'Farm_soainfra' > 'SOA' > 'soa-infra (soa_server1)' > 'default', with 'ShipDebitRequest [1.0]' selected. The main content area shows the 'ShipDebitRequest [1.0]' composite details. At the top, it indicates 'Running Instances: 0', 'Total: 0', and 'Active: 0'. Below this, there are tabs for 'Dashboard', 'Instances', 'Faults and Rejected Messages', 'Unit Tests', and 'Policies'. The 'Dashboard' tab is active, showing sections for 'Recent Instances', 'Recent Faults and Rejected Messages', 'Component Metrics', and 'Services and References'. The 'Services and References' section contains a table with the following data:

Name	Type	Usage	Faults	Total Messages	Average Processing Time (sec)
shpdrrequest_client_ep	Web Service	Service	0	0	0.000
ReadPayload	JCA Adapter	Reference	0	0	0.000
OZF_SD_REQUEST_PUB	Web Service	Reference	0	0	0.000

4. Click the Policies tab and notice that the 'oracle/wss_username_token_service_policy' policy you attached to the OZF_SD_REQUEST_PUB service binding earlier at the design time is now displayed here.

Policies Tab to Verify the Selected Security Policy



ShipDRequest [1.0] | Logged in as weblogic | Host | Page Refreshed Sep 19, 2011 12:06:51 PM PDT

Running Instances 0 | Total 4 | Active | Retire ... | Shut Down ... | Test | Settings ... | Related Links

Dashboard | Instances | Faults and Rejected Messages | Unit Tests | **Policies**

You can view and manage the list of policies attached to the web service bindings and components of this SOA composite application. Click 'Attach To/Detach From' to update the list of attached policies.

Policy Name	Attached To	Policy Referen Status
oracle/wss_username_token_client_policy	OZF_SD_REQUEST_PUB	Disable

5. In the ShipDebitRequest [1.0] home page, click **Test**.
6. The Test Web Service page for initiating an instance appears.

Note: If the WS-Security credentials are not entered at design time, you can enter the credentials at runtime by selecting the WSS Username Token option in the Security section at the top of the Request tab. Enter 'operations' in the Username field and the associated password for the user 'operations' in the Password field.

Request Tab to Enter the WS-Security Credentials



Request | Response

Security

WSS Username Token HTTP Basic Auth Custom Policy None

* Username operations Password *****

You can specify the following fields as XML payload data to use in the Input Arguments section:

Input Arguments Region

Name	Type	Value
* payload	payload	
* request_number	string	BPEL-1
* description	string	ShipDebitRequest
* req_max_qty	decimal	100

- request_number: Enter an 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.

Click **Test Web Service** to initiate the process.

The test results appear in the Response tab upon completion.

7. Verifying SOAP Response in the Console

In the Response tab page, click the Launch Message Flow Trace link to view the result of synchronous composite application. The Flow Trace page is displayed.

In the Trace section, verify that the ShipDebitRequest and OZF_SD_REQUEST_PUB components have a Completed state indicating that the application processed successfully.

You can check the Faults section to see if any error occurred during the test.

8. Click your BPEL service component instance link (such as ShipDebitRequest) to display the Instances page where you can view the implementation details of the BPEL activities in the Audit Trail tab.

Click the Flow tab to check the BPEL process flow diagram. Click an activity of the process diagram to view the activity details and flow of the payload through the process.

9. 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.

10. 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 page for the created request. Verify the details.

Invoking an Asynchronous Web Service from a SOA Composite Application with BPEL Process

SOA Composite Application with BPEL Process Scenario

This example explains the creation of a BPEL process to invoke a method `TESTUSERNAME` of asynchronous service `FND_USER_PKG` deployed on Oracle SOA Suite.

When the service has been successfully processed after deployment, the `TESTUSERNAME` method returns a positive number asynchronously if the user name passed as an input argument exists in Oracle E-Business Suite. If the user name does not exist, then number 0 is returned instead.

Searching and Recording a WSDL URL

Apart from the required tasks mentioned above, an integration developer needs to ensure that the service has been generated with the support of asynchronous operation pattern. This is achieved by selecting the Asynchronous interaction pattern checkbox in the Interaction Pattern table for the `TESTUSERNAME` operation of `FND_USER_PKG` service in integration Repository, and followed by clicking the **Generate** button to generate the service with the asynchronous operation. Once the service has been successfully generated, it needs to be deployed first before you copy the deployed WSDL URL to create a BPEL client at design time.

SOAP Web Service Tab with a Deployed Asynchronous WSDL URL

The screenshot displays the Oracle Integration Repository interface for the 'User' PLSQL Interface. The interface is titled 'Integration Repository > PLSQL Interface : User' and includes navigation buttons for 'Browse', 'Search', and 'Printable Page'. Key metadata includes: Internal Name: FND_USER_PKG, Type: PL/SQL, Product: Application Object Library, Status: Active, and Business Entity: User. The 'SOAP Web Service' tab is selected, showing a 'Deployed | Active' status and a 'Log Configuration Disabled' option. Under 'Service Operations', there are buttons for 'Retire', 'Undeploy', and 'Reset'. A table lists various service operations with columns for 'Display Name', 'Internal Name', 'Synchronous', 'Asynchronous', and 'Grant'. The 'Test User Name' operation is marked as asynchronous. Below the table, there is a 'Web Service Security' section with 'Authentication Type' set to 'Username Token'.

Display Name	Internal Name	Synchronous	Asynchronous	Grant
User	FND_USER_PKG	<input type="checkbox"/>	<input type="checkbox"/>	
Set Old Person Party Id	SET_OLD_PERSON_PARTY_ID	<input type="checkbox"/>	<input type="checkbox"/>	
Set Old User GUID	SET_OLD_USER_GUID	<input type="checkbox"/>	<input type="checkbox"/>	
Test User Name	TESTUSERNAME	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
User Form LDAP Wrapper Update User	FORM_LDAP_WRAPPER_UPDATE_USER	<input type="checkbox"/>	<input type="checkbox"/>	
Validate User Name	VALIDATE_USER_NAME	<input type="checkbox"/>	<input type="checkbox"/>	

Client Side Setup Tasks

Oracle E-Business Suite Integrated SOA Gateway asynchronous services have two port types. Each port type performs a one-way operation.

- One port type initiates the asynchronous process.
Example of this kind of port type can be 'TESTUSERNAME_ASYNCH'.
- The other one calls back the client with the asynchronous response.
Example of this kind of port type can be 'TESTUSERNAME_ASYNCH_RESPONSE'.

Oracle E-Business Suite Integrated SOA Gateway uses Web Services Addressing (WS-Addressing) to provide transport-neutral mechanisms to address and correlate web services and messages. WS-Addressing is a public specification and is the default correlation method supported by Oracle E-Business Suite Integrated SOA Gateway. It uses simple object access protocol (SOAP) headers for asynchronous message

correlation. Messages are independent of the transport or application used. For more WS-Addressing information, see <http://www.w3.org/Submission/ws-addressing/> for details.

To ensure that the asynchronous operation works properly, a Web Service Client needs to specify:

- Endpoint location (in `wsa:ReplyTo` header): The Reply-To address specifies the location at which a client that also implements Callback port-type is listening for a callback message.
- Conversation ID (in `wsa:MessageID` header): The MessageID uniquely identifies a message for the Web Service Client. The same MessageID is returned by Oracle E-Business Suite Integrated SOA Gateway in `wsa:RelatesTo` header in the callback message.

Establishing an Asynchronous Service Process Flow

To better understand how the asynchronous service will work, the following design-time tasks are included in this chapter:

1. Create a SOA Composite Application, page 3-49
Use this step to create a new SOA Composite application using an Asynchronous BPEL 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 to the request application.
2. Add a Partner Link, page 3-51
Use this step to create a partner link using the PL/SQL API FND_USER_PKG exposed as web service.
3. Add an Invoke activity, page 3-52
Use this step to configure an Invoke activity to invoke the service.
4. Add a Receive activity, page 3-53
Use this step to configure a Receive activity to receive the response message from the asynchronous operation and pass it on to the second Assign activity.
5. Add Assign activities, page 3-55
Use this step to configure two Assign activities in order to pass a user name from the client as an input value to the Invoke activity, as well as to pass the response message from the Receive activity. The response details will be written as an output file on the server where the BPEL process is deployed.

For general information and how to create SOA composite applications using BPEL process service component, see the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite* for details.

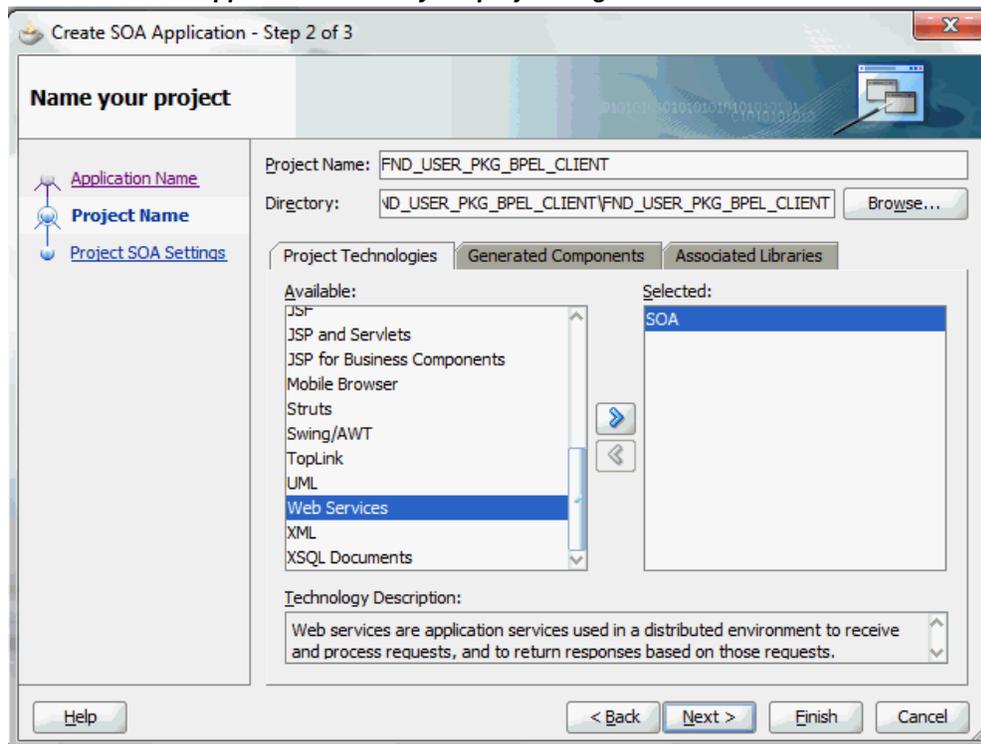
Creating a SOA Composite Application with BPEL Process

Use this step to create a new SOA Composite Application that will contain various BPEL process activities.

To create a new SOA Composite Application with BPEL project:

1. Open Oracle JDeveloper.
2. Click **New Application** in the Application Navigator.
The "Create SOA Application - Name your application" page is displayed.
3. Enter an appropriate name for the application in the Application Name field and select **SOA Application** from the Application Template list.
Click **Next**. The "Create SOA Application - Name your project" page is displayed.
4. Enter an appropriate name for the project in the Project Name field. For example, FND_USER_PKG_BPEL_CLIENT.

The Create SOA Application - Name your project Page



5. In the Project Technologies tab, select 'Web Services' and ensure that **SOA** is selected from the Available technology list to the Selected technology list.
Click **Next**. The "Create SOA Application - Configure SOA settings" page is displayed.
6. Select **Composite With BPEL Process** from the Composite Template list, and then click **Finish**.
You have created a new application, and a SOA project. This automatically creates a SOA composite.
The Create BPEL Process page is displayed.
7. Leave the default **BPEL 1.1 Specification** selection unchanged. This creates a BPEL project that supports the BPEL 1.1 specification.
Enter an appropriate name for the BPEL process in the Name field. For example, `TESTUSERNAME_ASYNC_BPELProcess`.
Select **Synchronous BPEL Process** in the Template field.
Select **requiresNew** from the Transaction drop-down list. Click **OK**.
A synchronous BPEL process is created with the Receive and Reply activities. The

required source files including `bpel` and `wSDL`, using the name you specified (for example, `TESTUSERNAME_ASYNC_BPELProcess.bpel` and `TESTUSERNAME_ASYNC_BPELProcess.wSDL`) and `composite.xml` are also generated.

8. Navigate to SOA Content > Business Rules and click `composite.xml` to view the composite diagram.

Double click on the `TESTUSERNAME_ASYNC_BPELProcess` component to open the synchronous BPEL process.

Adding a Partner Link

Use this step to configure a BPEL process by writing the response received from the asynchronous operation to a file.

To add a Partner Link:

1. In Oracle JDeveloper, place your mouse in the Partner Links area and right click to select **Create Partner Link...** from the pull-down menu. Alternatively, you can drag and drop **Partner Link** from the **BPEL Constructs** list into the right Partner Link swim lane of the process diagram.

The Create Partner Link window appears.

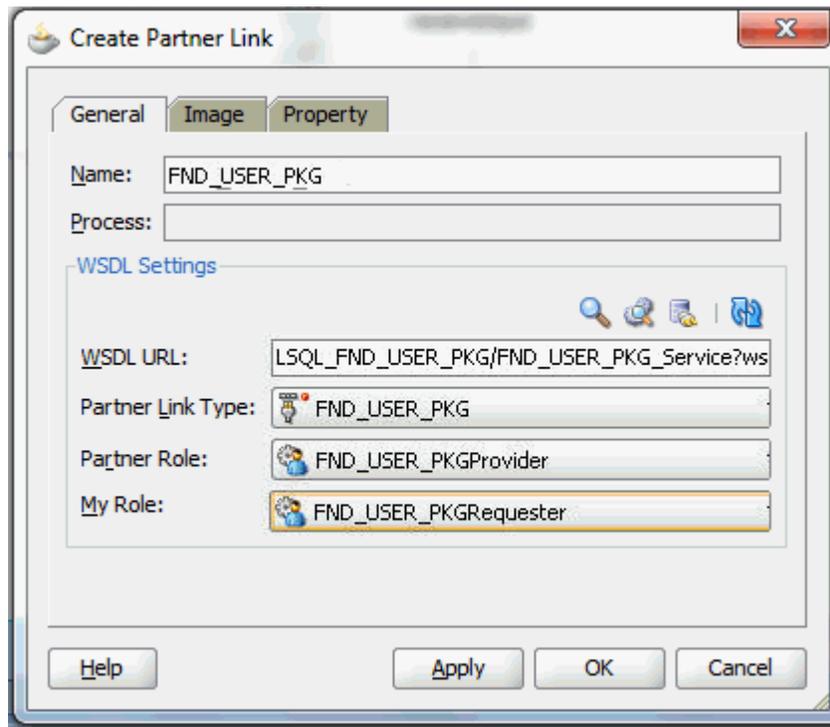
2. Copy the WSDL URL corresponding to the `FND_USER_PKG` service that you recorded earlier from the Integration Repository, and paste it in the WSDL File field. Press the **[Tab]** key.

3. Enter the partner link name as `FND_USER_PKG`.

Select the following values in the WSDL Settings region:

- Partner Link Type: `FND_USER_PKG`
- Partner Role: `FND_USER_PKGProvider`
- My Role: `FND_USER_PKGRequester`

Create Partner Link



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

Partner Link `FND_USER_PKG` is added to the Partner Links section in the BPEL process diagram.

Save your changes by selecting **File > Save All**.

Adding an Invoke Activity

Use this step to configure an **Invoke** activity for File Adapter to write the response to an output file.

To add an Invoke activity for WriteFile Partner Link:

1. In Oracle JDeveloper, expand the **BPEL Constructs** from the Component Palette. Drag and drop the **Invoke** activity into the center swim lane of the process diagram after the **receiveInput** activity.
2. Link the **Invoke** activity to the `FND_USER_PKG` 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

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

Select **Global Variable** and click **OK**.

Please note that we have selected asynchronous function.

4. 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 a Receive Activity

Use this step to configure a Receive activity to receive the response message from the asynchronous operation and pass it on to the second Assign activity.

1. In Oracle JDeveloper, expand the **BPEL Constructs** from the Component Palette. Drag and drop the **Receive** activity into the center swim lane of the process diagram after the **Invoke** activity.
2. Link the **Receive** activity to the `FND_USER_PKG` partner link. The Receive activity will receive the response message from the partner link. The Edit Receive dialog box appears.
3. Enter a name for the Receive activity. Select `TESTUSERNAME_ASYNCH_RESPONSE` from the Operation drop-down list.
4. Click the **Create** icon next to the **Variable** field to create a new variable. The Create Variable dialog box appears.

Select **Global Variable**, and then enter a name for the variable. You can accept the default name (such as `TESTUSERNAME_ASYNCH_Receive_TESTUSERNAME_ASYNCH_RESPONSE_InputVariable`). Click **OK** to return to the Edit Receive dialog box.

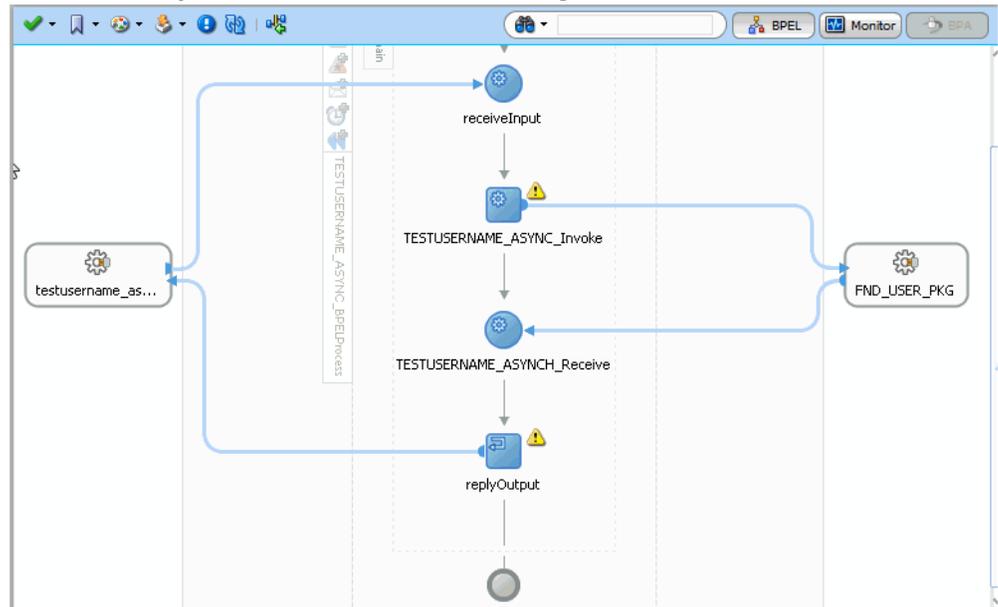
Edit Receive Dialog

The screenshot shows the 'Edit Receive' dialog box with the following configuration:

- Name:** TESTUSERNAME_ASYNCH_Receive
- Conversation ID:** (empty)
- Create Instance
- Interaction Type:** Partner Link
- My Role Web Service Interface:** (empty)
- Partner Link:** FND_USER_PKG
- Operation:** TESTUSERNAME_ASYNCH_RESPONSE
- Variable:** 1_TESTUSERNAME_ASYNCH_RESPONSE_InputVariable

5. Click **Apply** and **OK** to finish configuring the Receive activity.
The Receive activity appears in the BPEL process diagram.

Receive Activity Added in the BPEL Process Diagram



Adding Assign Activities

This step is to configure two **Assign** activities to pass a user name from the client as an input value to the Invoke activity, as well as to pass the response message from the Receive activity. The response details will be written as an output file on the server where the BPEL process is deployed.

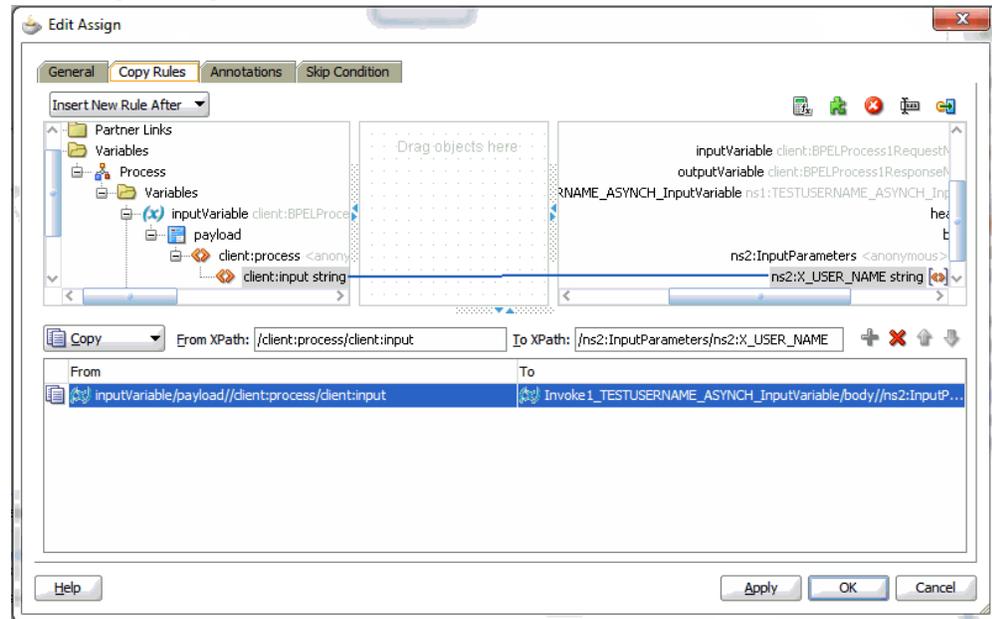
To enter the first Assign activity to pass a user name from the client as an input value to the Invoke activity:

1. Add the **Assign** activity by dragging and dropping the **Assign** activity from the **BPEL Constructs** in the Component Palette, between the **receiveInput** activity and the **Invoke** activity.
2. Double-click the **Assign** activity to access the Edit Assign dialog box.
Enter Input_to_Service as the name of the first **Assign** activity.
3. Select the Copy Rules tab and expand the source and target trees:
 - In the From navigation tree, navigate to **Variables > Process > Variables > InputVariable > payload > client:process > client:input**. The From XPath field is also displayed.
 - In the To navigation tree, navigate to **Variables > Process > Variables > TESTUSERNAME_ASYNCH_Invoke_TESTUSERNAME_ASYNCH_InputVariable > body > ns2:InputParameter** and select **ns2: X_USER_NAME**. The To

XPath field is also displayed.

4. Drag the source node (client:input) to point to the target node (ns2:X_USER_NAME) that you just identified in the previous step. This creates a line that connects the source and target nodes. The copy rule is displayed in the From and To sections at the bottom of the Edit Assign dialog box.

Edit Assign Dialog



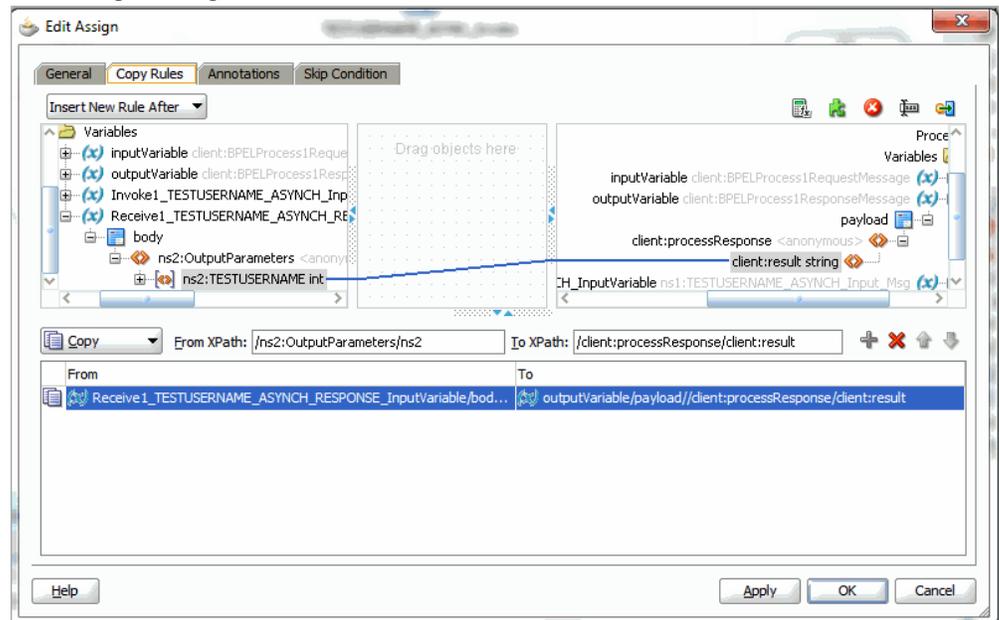
5. Click **Apply** and then **OK** to complete the configuration of the first **Assign** activity.

To enter the second Assign activity to pass the response message from the Receive activity:

1. Add the second **Assign** activity by dragging and dropping the **Assign** activity from the **BPEL Constructs** in the Component Palette, between the **Receive** activity and the **replyOut** activity.
2. Double-click the **Assign** activity to access the Edit Assign dialog box.
Enter **Service_Response** as the name of the second **Assign** activity.
3. Select the Copy Rules tab and expand the source and target trees:
 - In the From navigation tree, navigate to **Variables > Process > Variables > TESTUSERNAME_ASYNCCH_Receive_TESTUSERNAME_ASYNCCH_InputVariable > body > ns2:OutputParameters** and select **ns2:TESTUSERNAME**. The From XPath field is also displayed.

- In the To navigation tree, select type Variable and then navigate to **Variables > Process > Variables > OutputVariable > payload > client:processResponse** and select **client:result**. The To XPath field is also displayed.
4. Drag the source node (ns2:TESTUSERNAME) to point to the target node (client:result) that you just identified in the previous step. This creates a line that connects the source and target nodes. The copy rule is displayed in the From and To sections at the bottom of the Edit Assign dialog box.

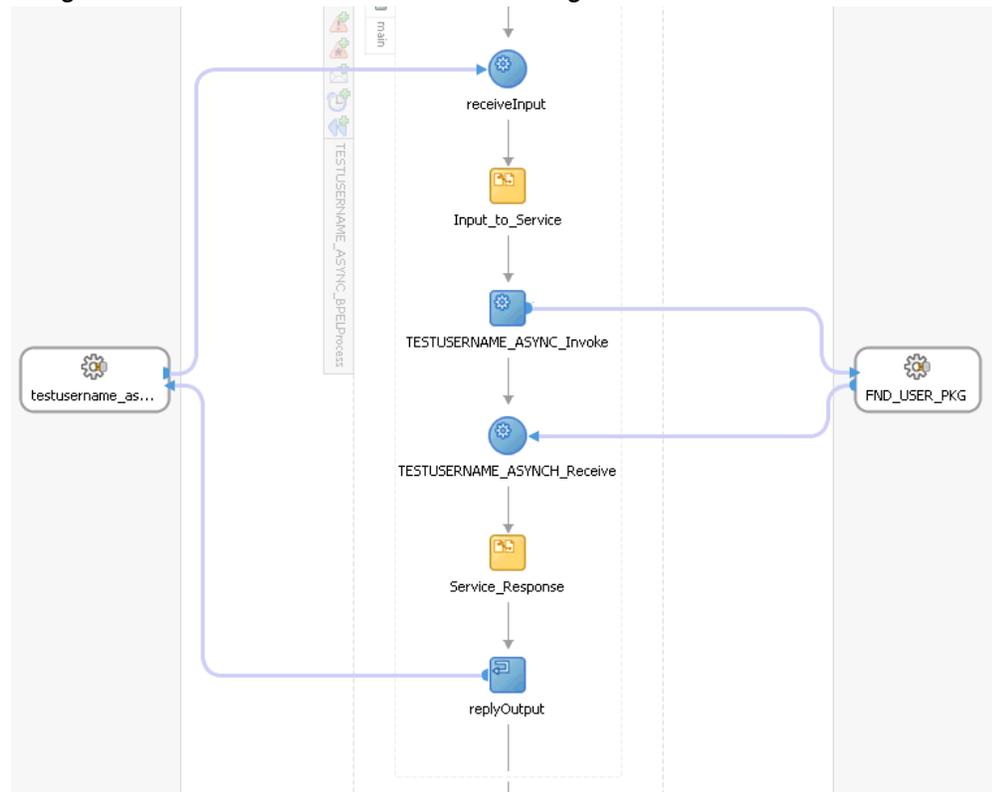
Edit Assign Dialog



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

The **Assign** activities appear in the BPEL process diagram.

Assign Activities Added in the BPEL Process Diagram

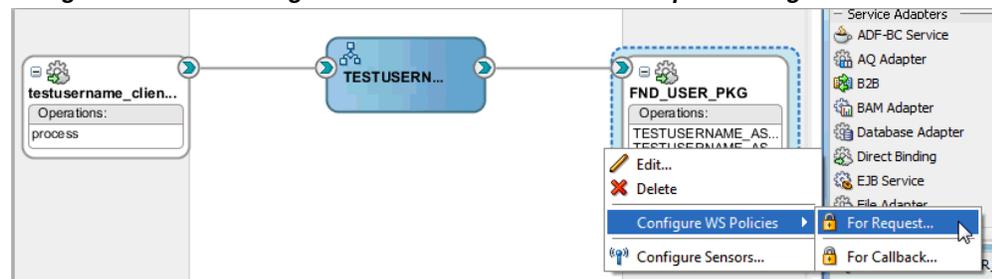


Configuring Web Service Policies

Use the following steps to add a security policy at design time:

1. Navigate to SOA Content > composite.xml. Right click on the FND_USER_PKG service and select "Configure WS Policies > For Request" from the drop-down list.

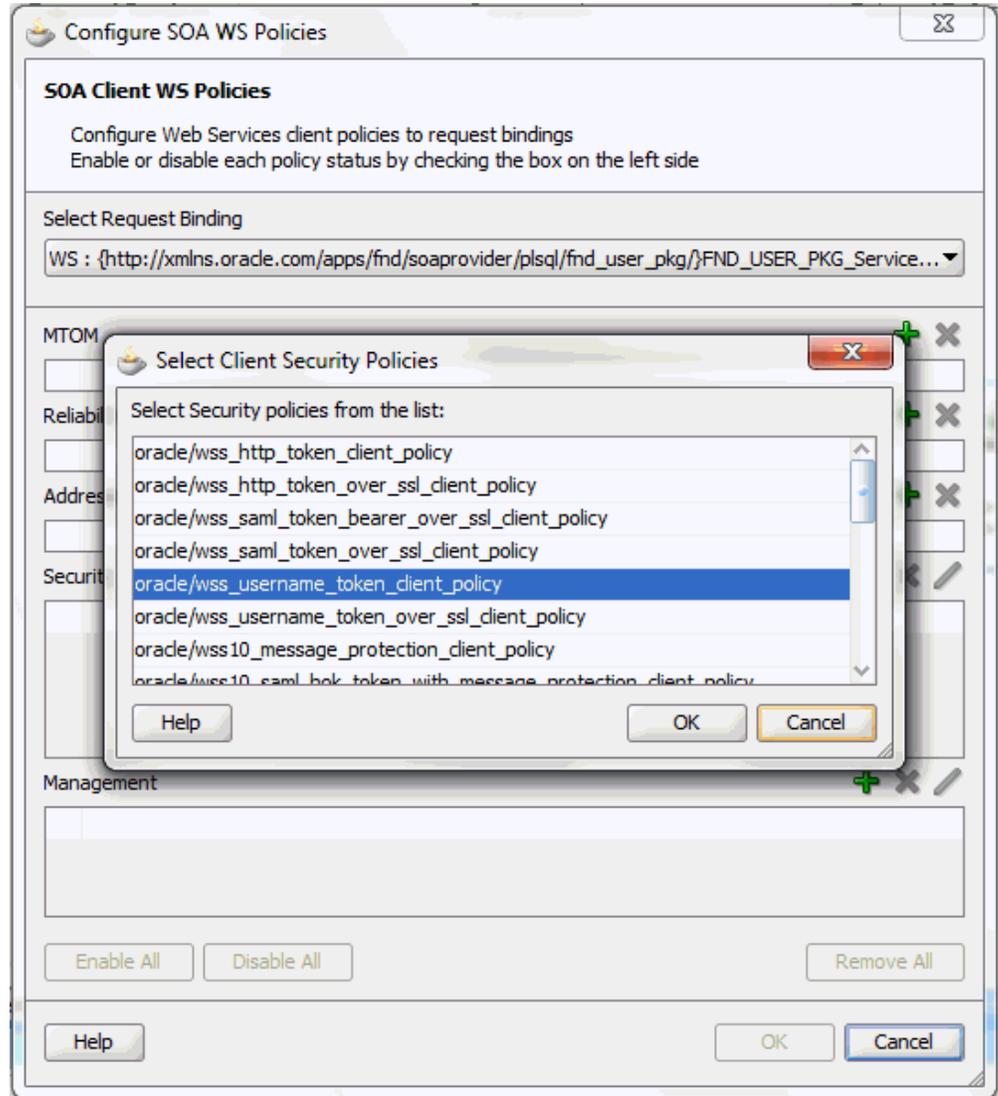
Navigation Path to Configure Web Service Policies in Composite Diagram



2. The Configure SOA WS Policies dialog appears. In the Security section, click the **Add** icon (+). The Select Server Security Policies

dialog appears.

Configure SOA WS Policies Dialog: Select Client Security Policies Dialog

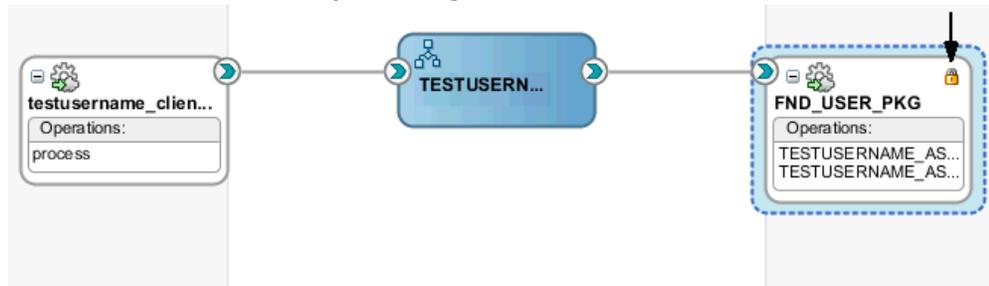


Select 'oracle/wss_username_token_service_policy' and click **OK**.

The attached security policy is shown in the Security section.

A lock icon appears in the FND_USER_PKG service of composite.xml indicating that a security policy has been successfully attached.

Lock Icon Shown in the Composite Diagram



3. Select the FND_USER_PKG service, and add the reference binding by selecting **View > Property Inspector** from the navigation menu. This displays the Property Inspector window for the FND_USER_PKG service component.

Note: This can also be done by directly adding the following reference binding in the Source tab of the composite.xml file for the FND_USER_PKG service:

```
<property name="oracle.webservices.auth.username"
  type="xs:string"
  many="false" override="may"
>operations</property>
<property name="oracle.webservices.auth.password"
  type="xs:string"
  many="false" override="may">password
</property>
```

Replace *password* with the actual password for the username 'operations'.

In the Properties section, click the **Add** icon (+) for binding properties. The Create Property dialog appears.

Enter 'oracle.webservices.auth.username' in the Name field and enter 'operations' as the value.

Create Property Dialog for Adding the Username Property

Create Property

Name: oracle.webservices.auth.username

Value: operations

Help OK Cancel

Click **OK**.

- Use the same approach by clicking the **Add** icon (+) again in the Properties section for binding properties. Enter 'oracle.webservices.auth.password' in the Name field. Enter the associated password for user 'operations' in the Value field.

Create Property Dialog for Adding the Password Property



Click **OK**.

Both selected property names and values appear in the Properties section.

Click the Source tab of the composite.xml and notice that the oracle.webservices.auth.username and oracle.webservices.auth.password property names and the associated values are added to the FND_USER_PKG reference.

Property Information Shown in Composite.xml



Deploying and Testing the SOA Composite with Asynchronous BPEL Process

To invoke the asynchronous TESTUSERNAME method from the BPEL client contained

in the SOA composite, the SOA composite needs to be deployed on the Oracle WebLogic managed server. This can be achieved using Oracle JDeveloper. Once the composite is deployed, it can be tested from the Oracle Enterprise Manager Fusion Middleware Control Console.

Prerequisites

Before deploying the SOA composite with BPEL process using Oracle JDeveloper, you must have established the connectivity between the design-time environment and the runtime server. For information on how to configure the necessary server connection, see *Configuring Server Connection*, page B-1.

Note: If a local instance of the WebLogic Server is used, start the WebLogic Server by selecting **Run > Start Server Instance** from Oracle JDeveloper. Once the WebLogic Admin Server "DefaultServer" instance is successfully started, the `<Server started in Running mode>` and `DefaultServer started` message in the `Running:DefaultServer` and `Messages` logs should appear.

Perform the following runtime tasks:

1. Deploy the SOA Composite Application with BPEL Process, page 3-62
2. Test the SOA Composite Application with BPEL Process, page 3-65

Deploying the SOA Composite with BPEL Process

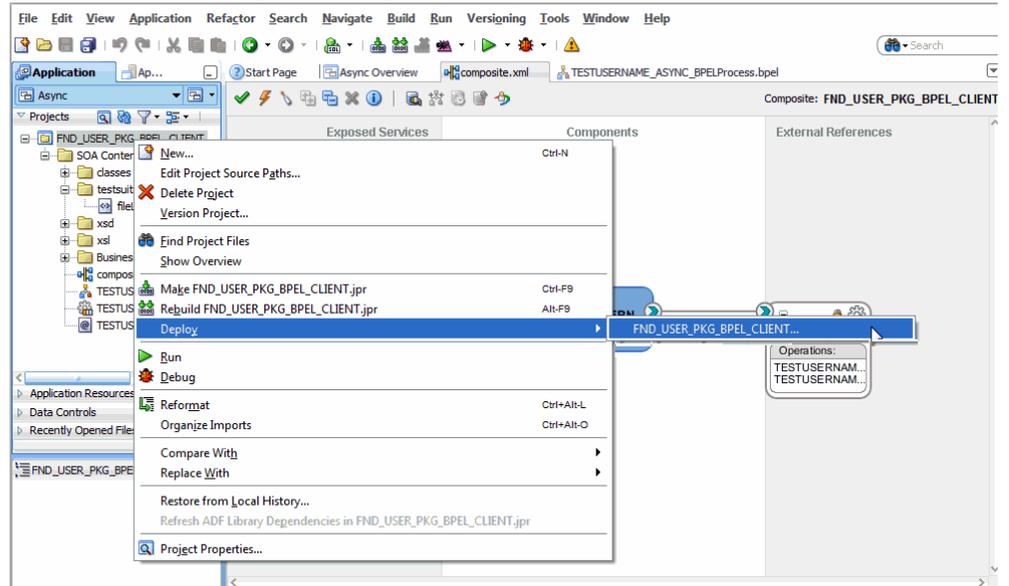
You must deploy the SOA composite with BPEL process that you created earlier before you can run it.

To deploy the SOA composite application:

1. In the Applications Navigator of JDeveloper, select the `FND_USER_PKG_BPEL_CLIENT` project.
2. Right-click the project and select **Deploy > [project name] > [serverConnection]** from the menu.

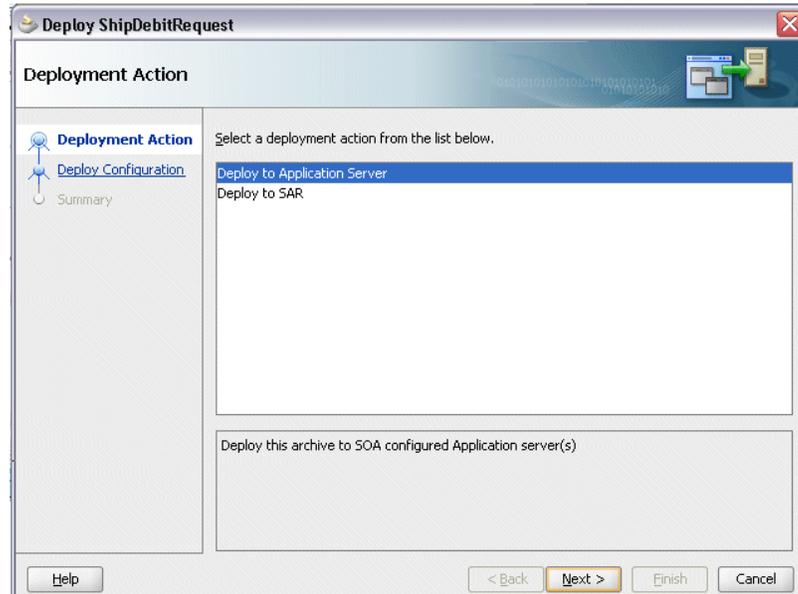
For example, you can select **Deploy > FND_USER_PKG_BPEL_CLIENT > SOAServer** to deploy the process if you have the connection set up appropriately.

Oracle JDeveloper Page to Deploy SOA Composite with BPEL Process



Note: If this is the first time to set up the server connection, then the Deployment Action dialog appears. Select 'Deploy to Application Server' and click **Next**.

Deployment Action Dialog



In the Deploy Configuration dialog, ensure the following information is selected before clicking **Next** to add a new application server:

- New Revision ID: 1.0
- Mark composite revision as default: Select this checkbox.
- Overwrite any existing composites with the same revision ID: Select this checkbox.

The steps to create a new Oracle WebLogic Server connection from JDeveloper are covered in Configuring Server Connection, page B-1.

3. In the Select Server dialog, select 'soa-server1' that you have established the server connection earlier. Click **Next**.
4. In the SOA Servers dialog, accept the default target SOA Server ('soa-server1') selection.
Click **Next** and **Finish**.
5. If you are deploying the composite for the first time from your Oracle JDeveloper session, the Authorization Request window appears. Enter the user name and password information specified during the Oracle SOA Suite installation. Click **OK**.

6. Deployment processing starts. Monitor deployment process and check for successful compilation in the SOA - Log window.

Verify that the deployment is successful in the Deployment - Log window.

Testing the SOA Composite Application

Once the BPEL process contained in the SOA composite application has been successfully deployed, you can manage and monitor the process from Oracle Enterprise Manager Fusion Middleware Control Console.

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

To test asynchronous operation in the Console:

1. Log in to Oracle Enterprise Manager Fusion Middleware Control Console (<http://<hostname>:<port>/em>) with the user name and password information specified during the installation.
2. From the Farm navigation pane, expand the SOA >soa-infra node in the tree to navigate through the SOA Infrastructure home page and menu to access your deployed SOA composite applications running on soa-infra managed server.
Click the FND_USER_PKG_BPEL_CLIENT [1.0] link.
3. In the FND_USER_PKG_BPEL_CLIENT [1.0] home page, click **Test**.
4. The Test Web Service page for initiating an instance appears. Enter the user name 'operations' as XML payload input data in the Input Arguments section.
Click **Test Web Service** to initiate the process.

Test Web Service Page: Request Tab

The screenshot shows the 'Test Web Service' interface. At the top, it indicates the user is logged in as 'weblogic|Host' and the page was refreshed on May 3, 2012, at 4:00:29 PM PDT. The main heading is 'Test Web Service' with a 'Test Web Service' button. Below this is a descriptive paragraph: 'Use this page to test any WSDL, including WSDLs that are not in the farm. To test a Web service, enter the WSDL and click Parse WSDL. When the page refreshes with the WSDL details, first select the Service, then select the Port, and then select the Operation that you want to test. Specify any input parameters, and click Test Web Service.'

The 'WSDL' field contains the URL 'http://.../soa-infra/services/default/FND_USER_PKG_BPEL_CLIENT/testusern'. Below it is a 'Parse WSDL' button and a link for 'HTTP Basic Auth Option for WSDL Access'. The 'Service' is 'testusername_async_bpelprocess_client_ep', the 'Port' is 'TESTUSERNAME_ASYNC_BPELProcess_pt', and the 'Operation' is 'process'. The 'Endpoint URL' is 'http://e.../soa-infra/services/default/FND_USER_PKG_BPEL'. There are 'Request' and 'Response' tabs, with 'Request' currently selected.

Under the 'Request' tab, there are expandable sections for 'Security', 'Quality of Service', 'HTTP Transport Options', and 'Additional Test Options'. The 'Input Arguments' section is expanded, showing a table with columns 'Name', 'Type', and 'Value'. A 'Tree View' dropdown is also present.

Name	Type	Value
* payload	payload	
* input	string	operations

5. The service invocation is successful and the response is shown in the Response tab.

Response Tab

The screenshot shows the 'Response' tab selected. It displays the following information: 'Test Status Request successfully received.', 'Response Time (ms) 3428', and a 'Tree View' dropdown. A message states 'A new composite instance was generated.' with a 'Launch Flow Trace' button. Below this is a table with columns 'Name', 'Type', and 'Value'.

Name	Type	Value
payload	payload	
result	string	2

The TESTUSERNAME operation returns a positive number asynchronously if the input user name passed as an argument exists in Oracle E-Business Suite. If the user name does not exist, then number 0 is returned instead.

6. Click **Launch Flow Trace** in the Response tab to open the Flow Trace page.

Flow Trace Page

Flow Trace
This page shows the flow of the message through various composite and component instances. ECID
Started **May 3, 2012 3:02:29 PM**

Faults (0)
Faults
Select a fault to locate it in the trace view.
Error Message Recovery Fault Time Fault Location Composite Instance
No faults found

Sensors (0)

Trace
Click a component instance to see its detailed audit trail.
Show Instance IDs

Instance	Type	Usage	State	Time	Composite Instance
testusername_async_bpelprocess_client_ep	Web Service	Service	Completed	May 3, 2012 3:02:29 PM	FND_USER_PKG_BPEL_CLIENT of
TESTUSERNAME_ASYNC_BPELProcess	BPEL Component		Completed	May 3, 2012 3:02:32 PM	FND_USER_PKG_BPEL_CLIENT of
FND_USER_PKG	Web Service	Reference	Completed	May 3, 2012 3:02:29 PM	FND_USER_PKG_BPEL_CLIENT of
FND_USER_PKG_Service	Web Service	Service	Completed	May 3, 2012 3:02:29 PM	_PLSQL_FND_USER_PKG of
Mediator	Mediator Component		Completed	May 3, 2012 3:02:32 PM	_PLSQL_FND_USER_PKG of
TESTUSERNAME_ASYNC	BPEL Component		Completed	May 3, 2012 3:02:32 PM	_PLSQL_FND_USER_PKG of
TESTUSERNAMEASYNCH	JCA Adapter	Reference	Completed	May 3, 2012 3:02:32 PM	_PLSQL_FND_USER_PKG of

7. In the Trace section, verify that the deployed 'TESTUSERNAME_ASYNC_BPELProcess' BPEL Component contained in the 'testusername_async_bpelprocess_client_ep' service, the FND_USER_PKG service, and the FND_USER_PKG_Service service all have a 'Completed' state indicating that the invocation is performed successfully.

You can also check the Faults section to see if any error occurred during the test.

8. Click the TESTUSERNAME_ASYNC link in the Flow Trace page to display the instance details in the Audit Trail tab.

Audit Trail Tab for a Component Instance

Flow Trace > Instance of TESTUSERNAME_ASYNCH

Instance of TESTUSERNAME_ASYNCH

This page shows BPEL process instance details.

Audit Trail | Flow | Sensor Values | Faults

Expand a payload node to view the details.

May 3, 2012 4:02:32 PM	Received "process" call from partner "TESTUSERNAME_ASYNCH"
View XML Document	
Transform_1	
May 3, 2012 4:02:32 PM	Updated variable "Invoke_1_TESTUSERNAMEASYNCH_InputVariable"
May 3, 2012 4:02:32 PM	Completed assign
Invoke_1	
May 3, 2012 4:02:32 PM	Started invocation of operation "TESTUSERNAMEASYNCH" on partner "TESTUSERNAMEASYNCH".
May 3, 2012 4:02:32 PM	Sending property "jca.apps.Username", value is "operations".
May 3, 2012 4:02:32 PM	Sending property "jca.apps.NLSLanguage", value is "".
May 3, 2012 4:02:32 PM	Sending property "jca.apps.SecurityGroup", value is "".
May 3, 2012 4:02:32 PM	Sending property "jca.apps.Responsibility", value is "".
May 3, 2012 4:02:32 PM	Sending property "jca.apps.RespApplication", value is "".
May 3, 2012 4:02:32 PM	Sending property "jca.apps.ORG_ID", value is "".
May 3, 2012 4:02:34 PM	Invoked 2-way operation "TESTUSERNAMEASYNCH" on partner "TESTUSERNAMEASYNCH".
<payload>	
<pre><messages> <Invoke_1_TESTUSERNAMEASYNCH_InputVariable> <part name="InputParameters"> <InputParameters> <X_USER_NAME>operations</X_USER_NAME> </InputParameters> </part> </Invoke_1_TESTUSERNAMEASYNCH_InputVariable> <Invoke_1_TESTUSERNAMEASYNCH_OutputVariable> <part name="OutputParameters"> <OutputParameters> <TESTUSERNAME>2</TESTUSERNAME> </OutputParameters> </part> </Invoke_1_TESTUSERNAMEASYNCH_OutputVariable> </messages></pre>	
Transform_2	
May 3, 2012 4:02:34 PM	Updated variable "outputVariable"
May 3, 2012 4:02:34 PM	Completed assign
callbackClient	
May 3, 2012 4:02:34 PM	Started invocation of operation "processResponse" on partner "TESTUSERNAME_ASYNCH".
May 3, 2012 4:02:34 PM	Invoked 1-way operation "processResponse" on partner "TESTUSERNAME_ASYNCH".
<payload>	
<pre><outputVariable> <part name="payload"> <OutputParameters> <TESTUSERNAME>2</TESTUSERNAME> </OutputParameters> </part> </outputVariable></pre>	
May 3, 2012 4:02:34 PM	BPEL process instance "20131" completed

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 the 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 3-69
- Invoking a REST Service Using Token Based Authentication and JSON Payload, page 3-79

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 user name 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

Application 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
<i>Responsibility</i>	responsibility_key (such as "SYSTEM_ADMINISTRATOR")

Element Name	Expected Value
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 invocation 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 3-70
2. Recording Resource Information from Deployed WADL, page 3-72
3. Creating a Project with a Java Class, page 3-73
4. Invoking a REST Service Using a Java Class, page 3-79

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 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.

- Enter 'FND_PROFILE' in the Internal Name field. Click **Go**.
Click the 'Profile Management APIs' interface name link to open the interface details page.
- In the REST Web Service tab, enter the following information:

Interface Details Page with REST Web Service Tab

The screenshot shows the 'Profile Management APIs' interface details page in the Oracle Integration Repository. The page is titled 'PLSQL Interface : Profile Management APIs' and includes a navigation bar with 'Integration Repository' and 'Administration' tabs. The main content area displays the following information:

- Internal Name:** FND_PROFILE
- Type:** PL/SQL
- Product:** Application Object Library
- Status:** Active
- Business Entity:** [User Profile](#)
- Online Help:** [See the related online help](#)
- Scope:** Public
- Interface Source:** Oracle

Below this information, there are tabs for 'Overview', 'SOAP Web Service', 'REST Web Service' (selected), and 'Grants'. The 'REST Web Service' tab contains the following configuration options:

- Service Alias:** fndProfileSvc
- REST Service Status:** Not Deployed
- Log Configuration:** Disabled
- Configure** button

The 'Service Operations' section contains a table with the following data:

Name	Internal Name	Get	Post	Grant
Profile Management APIs		<input type="checkbox"/>	<input type="checkbox"/>	
Put Profile	PUT	<input type="checkbox"/>	<input type="checkbox"/>	
Get Profile	GET	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="button" value="Grant"/>
Get Profile Value	VALUE	<input type="checkbox"/>	<input type="checkbox"/>	

Below the table, there is a tip: 'TIP To apply any changes in Operation, Undeploy the service.' and a 'Table Diagnostics' button. The 'REST Service Security' section includes the following configuration options:

- Authentication Type:**
 - HTTP Basic
 - Security Token
- Tip:** Use [Login Service](#) to obtain Security Token for given user credentials.
- Deploy** button

At the bottom of the page, there are 'Browse', 'Search', and 'Printable Page' buttons, and a footer with copyright information: 'Copyright (c) 1998, 2021, Oracle and/or its affiliates. All rights reserved.' and links for 'About this Page' and 'Privacy Statement'.

- Service Alias: FndProfileSvc
The alias will be displayed as the service endpoint in the WADL and schema for the selected method or operation which is 'Get Profile' in this example.
- In the Service Operations region, select the "Post" checkbox for the 'Get Profile' service operation.
The selected method will be exposed as a REST service operation.

- In the REST Service Security region, ensure that at least one authentication type is selected.

5. 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.

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.

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

The screenshot shows the REST Web Service configuration page for the service 'fndProfileSvc'. The 'REST Service Status' is 'Deployed', and the 'View WADL' link is highlighted with a red box. Below the status, there is a 'Service Operations' section with a table of operations. The 'REST Service Security' section shows 'HTTP Basic' authentication selected.

Service Alias: fndProfileSvc
REST Service Status: Deployed | [View WADL](#) | Log Configuration Disabled

Service Operations

Name	Internal Name	Get	Post	Grant
Profile Management APIs				
Put Profile	PUT			
Get Profile	GET	✓		🔒
Get Profile Value	VALUE			

REST Service Security

*Authentication Type: HTTP Basic Security Token

Tip: Use [Login Service](#) to obtain [Security Token](#) for given user credentials.

The following WADL description appears:

```

<application xmlns:tns="http://xmlns.oracle.
com/apps/fnd/soapprovider/plsql/rest/fnd_profile/"
  xmlns="http://w3.org/2001/XMLSchema" xmlns:xsd="http://www.w3.
org/2001/XMLSchema"
  xmlns:tns1="http://xmlns.oracle.com/apps/fnd/rest/fndProfileSvc/get/"
  name="FND_PROFILE"
  targetNamespace="http://xmlns.oracle.
com/apps/fnd/soapprovider/plsql/rest/fnd_profile/">
<grammars>
  <include xmlns="http://www.w3.org/2001/XMLSchema"
    href="http://<hostname>:<port>/webservices/rest/FndProfileSvc/?
XSD=GET_SYNCH_TYPEDEF.xsd" />
</grammars><resources base="http://<hostname>:
<port>/webservices/rest/FndProfileSvc/"><resource path="/get/">
  <method id="GET" name="POST">
    <request>
      <representation mediaType="application/xml" type="tns1:GET_Input"
/>
      <representation mediaType="application/json" type="tns1:GET_Input"
/>
    </request>
    <response>
      <representation mediaType="application/xml" type="tns1:GET_Output"
/>
      <representation mediaType="application/json" type="tns1:GET_Output"
/>
    </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/FndProfileSvc/">

This information http://<hostname>:

<port>/webservices/rest/FndProfileSvc 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

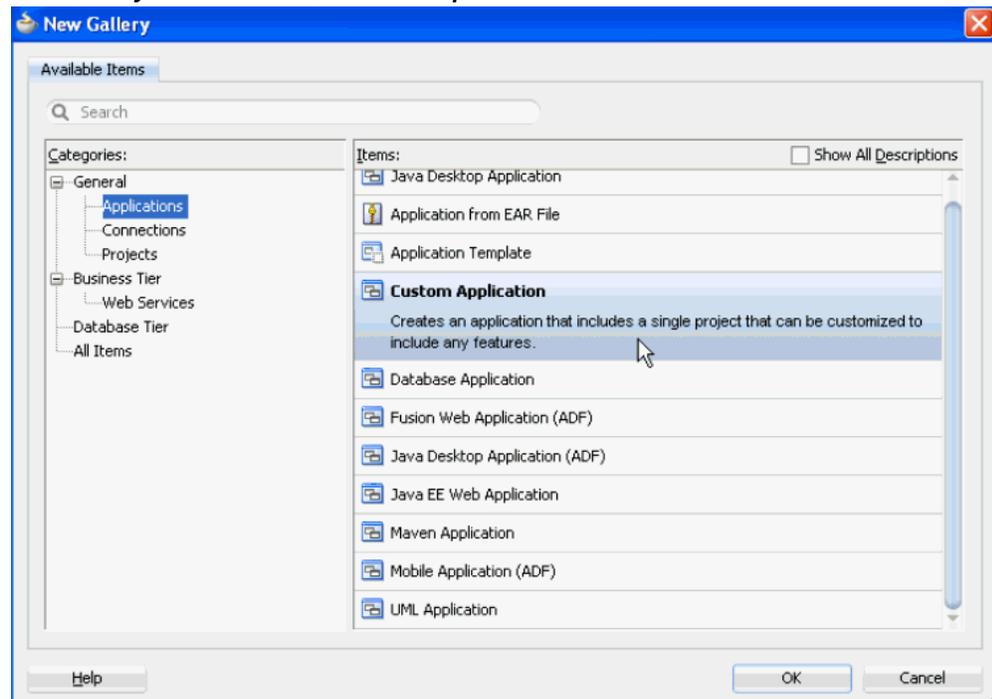
This section describes how to create a project with a Java class that will be used to invoke the FND_PROFILE REST service.

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**.

New Gallery Window in Oracle JDeveloper



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.

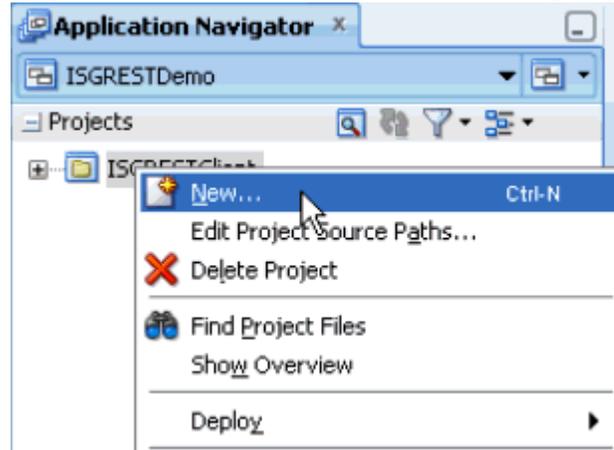
Click **Next**.

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

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

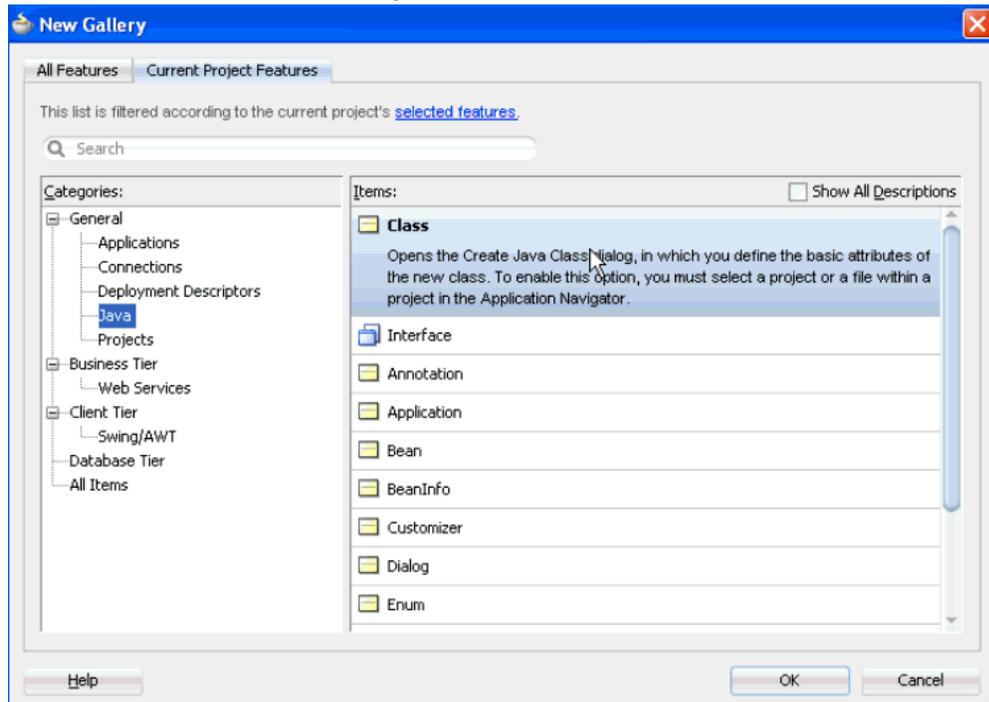
6. Select and right-click on the project name you just created in the Application Navigator and choose **New** from the drop-down selection menu.

Drop-Down Menu to Select New from an Existing Project



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

Java Class Selected in New Gallery Window



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/soapprovider/plsql/rest/fnd_profile/get/\\"
+ "      xmlns:ns1=\\"http://xmlns.oracle.
com/apps/fnd/soapprovider/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(URLConnection 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, "<EBS username>", "<password>");
    } 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 TLS-enabled environment. Additionally, you need to import the TLS certificate into your client JVM's keystore.

10. Replace `<server hostname>:<port>`, `<EBS username>`, and `<password>` 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 run 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 and verify this process and check for successful compilation 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/FndProfileSvc/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 **FndProfileSvc** entered earlier during the 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 Login security 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.

Important: To secure each user session, a profile option "FND: Authn Service Token Scope (FND_AUTHN_SRVC_TOKEN_SCOPE)" is introduced in the January 2023 Critical Patch Update with the default value "Header Only" at the site level. The recommended value for the profile option will set the ICX cookie only in the response HTTP header of the Login security service. It will not return the access token details in the response payload.

If you have custom code that uses Login security service to retrieve the access token details from the response payload, then you must change the profile value to "Header and Body" *temporarily*. It will provide the access token details in the response payload in addition to the ICX cookie in the response HTTP header. For security and privacy purposes, we recommend that you modify your custom code to retrieve the access token details from the cookie in the response header as soon as possible. After your custom code has been updated to meet recommended security standards, you should change the "FND: Authn Service Token Scope (FND_AUTHN_SRVC_TOKEN_SCOPE)" profile value to "Header Only" at the site level.

In this example, user name information to be tested is passed in a JSON-based payload for the REST service invocation. When the service has been successfully invoked, 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:

Obtaining a Needed Library

To successfully invoke the REST service with payload in JSON format, you need to obtain the jersey-bundle_1.0.0.0_1-1-5-1.jar library available at the <\$COMMON_TOP>/java/lib directory.

Setting Variables in RESTHeader for an HTTP Request

In REST services, application 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 3-69.

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 3-81
2. Recording the Deployed WADL URL, page 3-83
3. Creating a Project with a Java Class, page 3-84
4. Invoking a REST Service Using a Java Class, page 3-95

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 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 run the search. Click the 'User' interface name link to open the interface details page.
4. In the REST Web Service tab, enter the following information:

Deploy a PL/SQL REST Service

The screenshot displays the Oracle Integration Repository Administration interface. At the top, it shows the breadcrumb 'Integration Repository > Administration'. The main heading is 'PLSQL Interface : User'. Below this, there are tabs for 'Browse', 'Search', and 'Printable Page'. The interface is divided into several sections:

- Metadata:** Internal Name: FND_USER_PKG, Type: PL/SQL, Product: Application Object Library, Status: Active, Business Entity: User, Online Help: See the related online help. Scope: Public, Interface Source: Oracle.
- REST Web Service:** Service Alias: fndMessageSvc, REST Service Status: Not Deployed, Log Configuration: Enabled. Buttons for 'Configure' and 'View Log' are present.
- Service Operations:** A table listing various operations with columns for Name, Internal Name, Get, Post, and Grant. The 'Test User Name' operation (Internal Name: TESTUSERNAME) has the 'Post' checkbox checked.
- REST Service Security:** Personalize "REST Service Security" section with 'Authentication Type' options: HTTP Basic (checked) and Security Token (checked). A tip suggests using 'Login Service' to obtain a Security Token.

At the bottom of the page, there are 'Browse', 'Search', and 'Printable Page' buttons, and a footer with copyright information and links for 'About this Page' and 'Privacy Statement'.

- Service Alias: fndMessageSvc
The alias will be displayed as the service endpoint in the WADL and schema for the selected method or service operation which is TESTUSERNAME in this example.
- In the Service Operations region, select the "Post" checkbox for the 'Test User Name' (TESTUSERNAME) service operation.
The selected method will be exposed as a REST service operation with the support for "Post" method.

- In the REST Service Security region, ensure that at least one authentication type is selected.
5. 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 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

Overview SOAP Web Service **REST Web Service** Log Configuration Enabled

Service Alias fndMessageSvc
REST Service Status Deployed **View WADL**

Service Operations

Name	Internal Name	Get	Post	Grant
Change User Name	CHANGE_USER_NAME			
Validate User Name	VALIDATE_USER_NAME			
Test User Name	TESTUSERNAME		✓	
Pre-validate EBS user	ISUSERACTIVE			
Create/Update User	LOAD_ROW			
Remove PII user information	REMOVE_PII_USER			

TIP To apply any changes in Operation, Undeploy the service.

Table Diagnostics

REST Service Security

*Authentication Type HTTP Basic Security Token

Tip: Use [Login Service](#) to obtain Security Token for given user credentials.

The following WADL description appears:

```

<application xmlns:tns="http://xmlns.oracle.
com/apps/fnd/soapprovider/plsql/rest/fnd_user_pkg/"
  xmlns="http://w3.org/2001/XMLSchema" xmlns:xsd="http://www.w3.
org/2001/XMLSchema"
  xmlns:tns1="http://xmlns.oracle.
com/apps/fnd/rest/fndMessageSvc/testusername/" name="FND_USER_PKG"
  targetNamespace="http://xmlns.oracle.
com/apps/fnd/soapprovider/plsql/rest/fnd_user_pkg/">
<grammars>
  <include xmlns="http://www.w3.org/2001/XMLSchema" href="http:
//<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:
TESTUSERNAME_Input"/>
      <representation mediaType="application/json" type="tns1:
TESTUSERNAME_Input"/>
    </request>
    <response>
      <representation mediaType="application/xml" type="tns1:
TESTUSERNAME_Output"/>
      <representation mediaType="application/json" type="tns1:
TESTUSERNAME_Output"/>
    </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:

```

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 java.util.StringTokenizer;

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.toLowerCase() + "=" +
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, String dbsid) throws Exception {

String rfUrl = baseUrl + "/login";
URL url = new URL(rfUrl);
// Obtaining connection to invoke login service.
URLConnection conn = (URLConnection) 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();
}
String value = readCookie(conn, dbsid);
if(value != null)
return (new String[] { dbsid, value });
else
return null;
}

public static String readCookie(URLConnection urlConn, String
key) {
int i = 1;
String hdrKey;
String hdrString;
String aCookie;
while ((hdrKey = urlConn.getHeaderFieldKey(i)) != null) {
if (hdrKey.equals("Set-Cookie")) {
hdrString = urlConn.getHeaderField(i);
StringTokenizer st = new StringTokenizer(hdrString, ",");
while (st.hasMoreTokens()) {
String s = st.nextToken();
aCookie = s;
int j = aCookie.indexOf("=");
if (j != -1) {
if(key.equalsIgnoreCase(aCookie.substring(0, j)))
return aCookie.substring(j + 1);
}
}
}
}
}

```

```

    }
    }
    i++;
    }
    return null;
    }

/**
 * This method reads response sent by the server and returns it in
a string
 * representation.
 */
private static String readHttpResponse(URLConnection 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 +
"/findMessageSvc/testusername/";
    String dbSid = "<sid_of_database>";
    // Get Access Token by invoking AOL Login Service
    String[] token = getAolToken(baseUrl, "sysadmin", "sysadmin",
dbSid);
    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 TLS-enabled environment. Additionally, you need to import the TLS certificate into your client JVM's keystore.

9. Replace `<server hostname>:<port>`, `<EBS username>`, and `<password>` with actual values in the code.

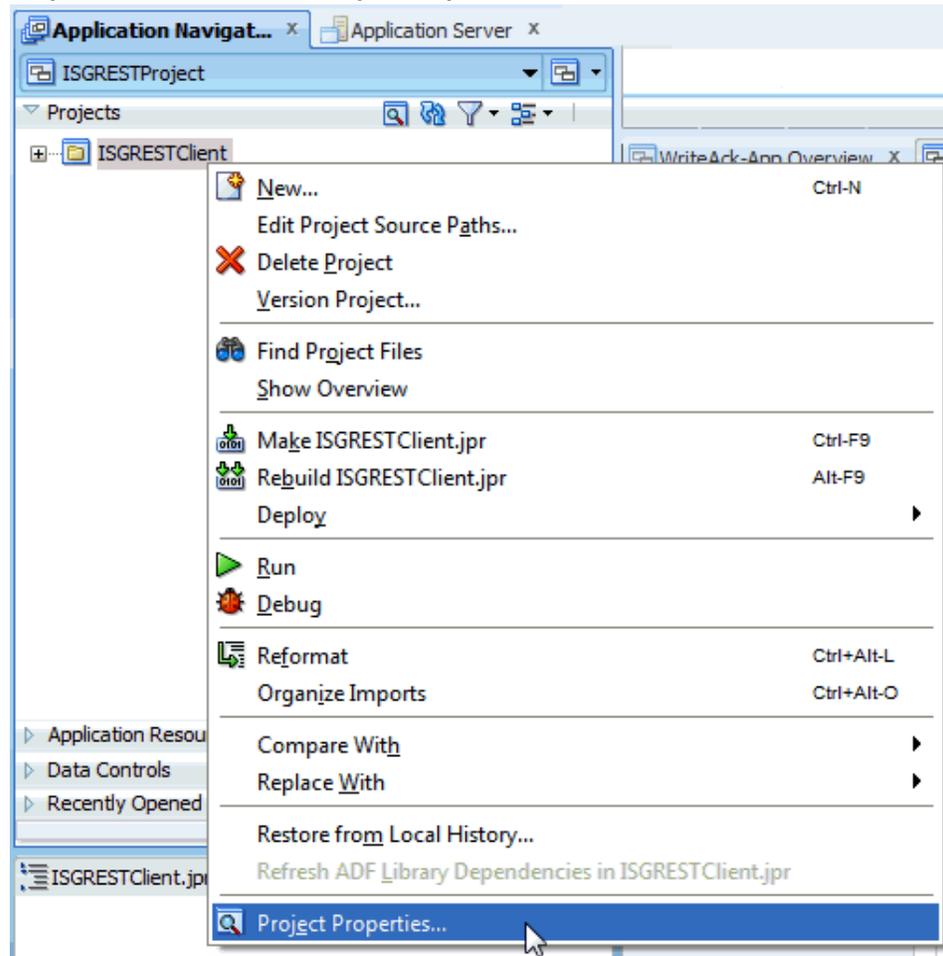
10. Replace `<sid_of_database>` with an actual value in the code.

11. 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.

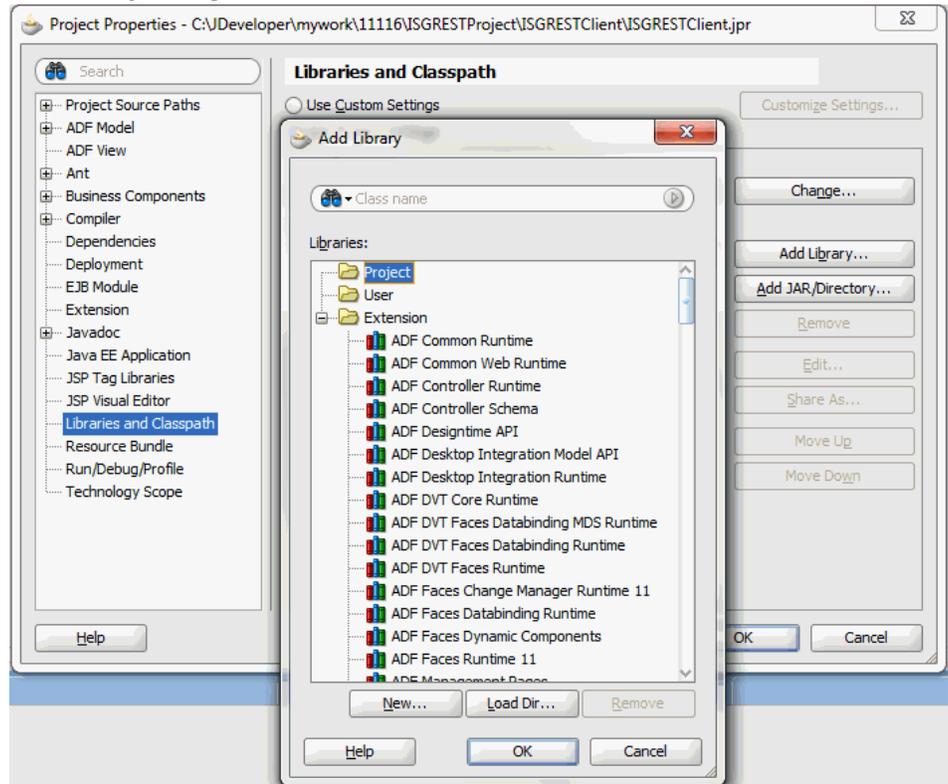
Drop-Down Menu to Select Project Properties



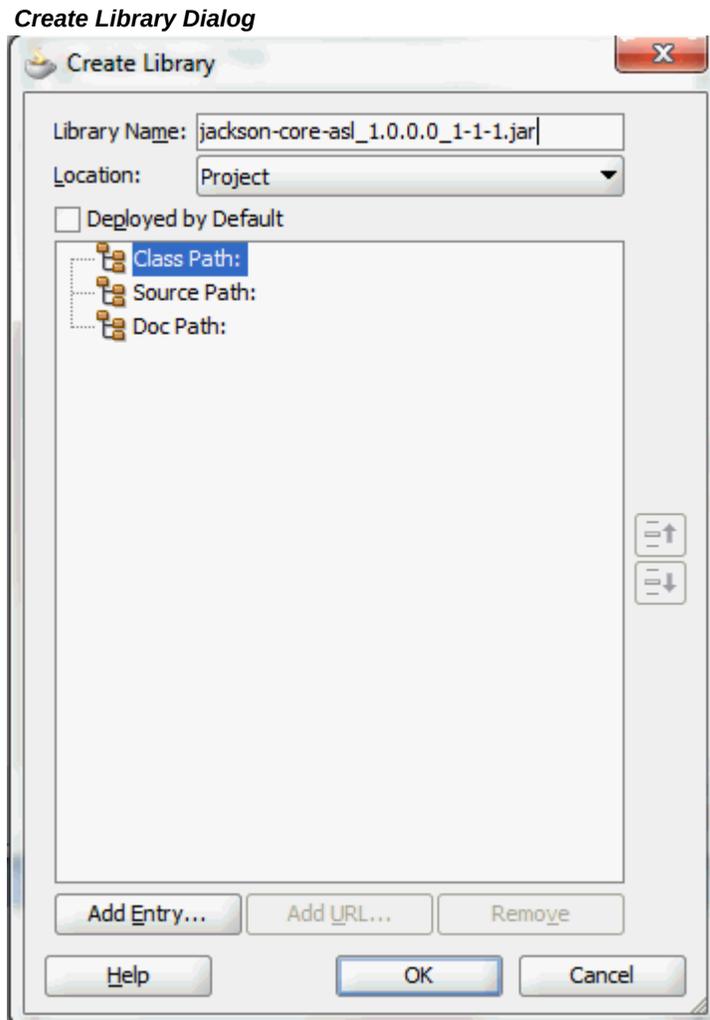
The Default Properties dialog box opens.

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

Add Library Dialog



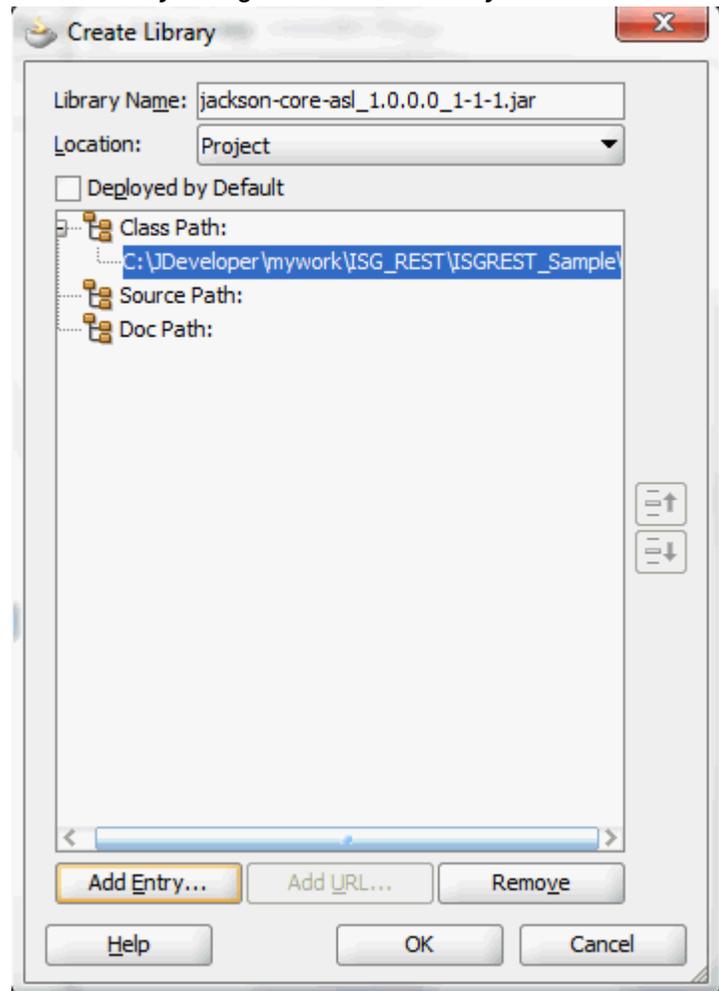
4. In the Add Library dialog box, select the Project folder and then click **New**. The Create Library dialog box opens.
5. In the Library Name field, enter 'jackson-core-asl_1.0.0.0_1-1-1.jar'.



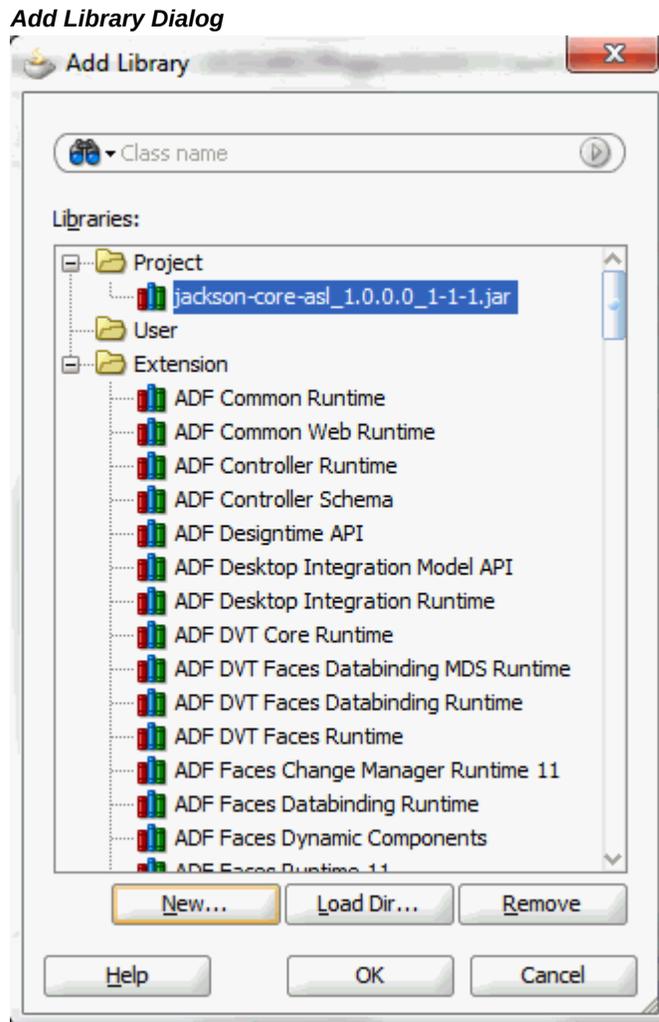
Click **Add Entry**. The Select Path Entry dialog box appears.

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.

Create Library Dialog with Selected Library Path



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



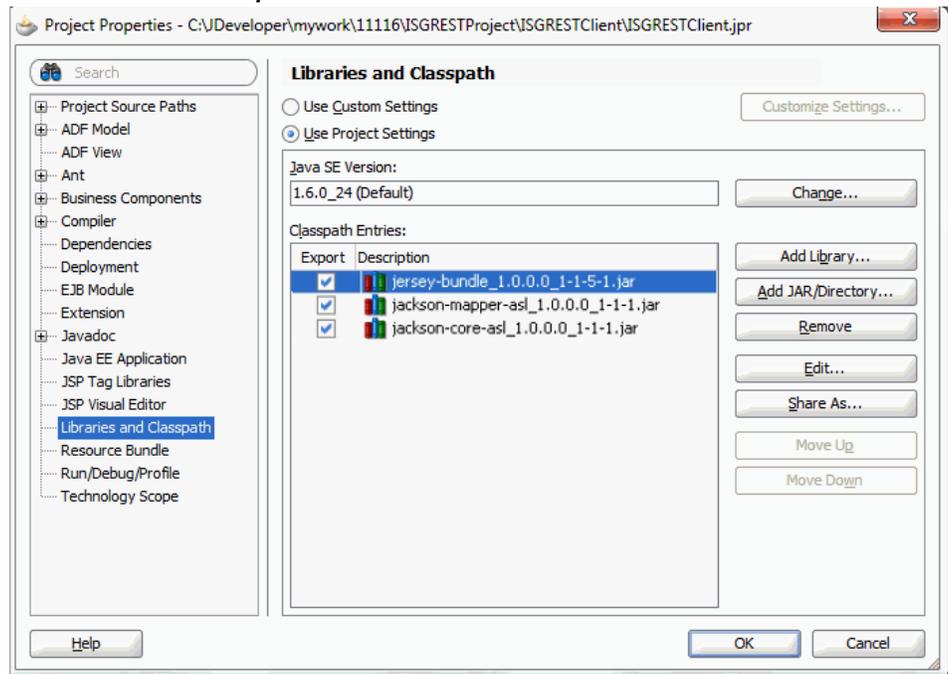
7. Repeat steps 4, 5, and 6 to add the following two jar files to the Project folder:

- 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.

Verification of the Required Libraries



12. 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 run 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 process is successfully run 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 - xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

- 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 the 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
  }
}
```

Using Java APIs 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**. Some Java class provides access to business logic governing the OA Framework-based components and pages. Such Java classes are called **Application Module Services** and are also categorized as a subtype of Java interface.

To locate these subtype of Java interfaces, perform a search by specifying "Interface Subtype" as the Category and "Java Bean Services" or "Application Module Services" as the Category Value. Both Java Bean Services and Application Module Services can be exposed as REST services only.

Once Java Bean Services or Application Module 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.

Service Invocation Examples

To better understand how to use Java APIs as REST services to fetch and use application data, this chapter includes service invocation examples for interface types of Java Bean Services and Application Module Services. Oracle JDeveloper 11g (11.1.1.6.0) is used in these examples to create projects with Java class and invoke the services.

- Invoking a Java Bean Service Using HTTP GET Method, page 4-2
- Annotating and Invoking a Custom Java Bean Service, page 4-15
- Invoking an Application Module Service Using Token Based Authentication and XML Payload, page 4-48

Invoking a Java Bean 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 user name and password information in the HTTP request header. The user credentials are authenticated and authorized by the REST service provider of ISG. After the validation, the `getRestInterface` service operation processes 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 the HTTP URL at runtime during the 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 4-8
5. Invoking a REST Service Using a Java Class, page 4-14

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 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**.

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:

REST Web Service Tab to Deploy a Java REST Service

Service Alias:

REST Service Status: Not Deployed

Log Configuration Disabled [Configure](#)

Service Operations

Display Name	Internal Name	GET	POST	Grant
Rest Service Locator	oracle.apps.fnd.rep.ws.service.EbsRestLocator	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
ADDGRANT	addGrant	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Grant
ADDGRANTS	addGrants	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Grant
GETOPERATIONS	getOperations	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Grant
LISTGRANTS	listGrants	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Grant
REVOKEGRANT	removeGrant	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Grant
getRestInterface	getRestInterface	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Grant
getRestInterfaces	getRestInterfaces	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Grant

REST Service Security

*Authentication Type HTTP Basic Security Token

Tip: Use [Login Service](#) to obtain Security Token for given user credentials.

[Deploy](#)

- Service Alias: servicelocator

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 checkboxes are preselected.

Please note that if a Java method is annotated with a specific HTTP method,

then the corresponding HTTP method checkbox is preselected for that method. The administrator can change the HTTP method checkbox 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 checkbox is automatically selected.

- In the REST Service Security region, ensure that at least one authentication type is selected.

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.

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 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 checkbox and then click **Create Grant**.

Interface Details Page with Grants Tab

Integration Repository Administration

Integration Repository >

Java Details : Rest Service Locator

[Browse](#) [Search](#) [Printable Page](#)

Internal Name oracle.apps.fnd.rep.ws.service.EbsRestLocator
Type Java
Product Application Object Library
Status Active

Scope Public
Interface Source Oracle
Interface Subtype Java Bean Services

Overview REST Web Service **Grants**

Select Object and [Create Grant](#) [Revoke Grant](#)

Select All | Select None

Select	Name ▲	Internal Name ▲	REST Service Operation ▲	Grant ▲
<input type="checkbox"/>	ADDGRANT	addGrant		
<input type="checkbox"/>	ADDGRANTS	addGrants		
<input type="checkbox"/>	GETOPERATIO...	getOperations		
<input type="checkbox"/>	LISTGRANTS	listGrants		
<input type="checkbox"/>	REVOKEGRANT	removeGrant		
<input checked="" type="checkbox"/>	getRestInterface	getRestInterface		
<input type="checkbox"/>	getRestInterfaces	getRestInterfaces		

5. In the Create Grants page, select "All User" 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

Integration Repository Administration

Integration Repository > Java Details : Rest Service Locator >

Create Grants

Cancel Create Grant

Selected Methods

Name	Internal Name
getRestInterface	getRestInterface

Grant All Selected

Grantee Type All Users

Copyright (c) 1998, 2015, Oracle and/or its affiliates. All rights reserved. Privacy Statement

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.

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>/webservicess/rest/servicelocator/?XSD=addgrant.xsd" xmlns="http:
//www.w3.org/2001/XMLSchema" />
  <include href="http://<hostname>:
<port>/webservicess/rest/servicelocator/?XSD=addgrants.xsd" xmlns="http:
//www.w3.org/2001/XMLSchema" />
  <include href="http://<hostname>:
<port>/webservicess/rest/servicelocator/?XSD=getoperations.xsd" xmlns="
http://www.w3.org/2001/XMLSchema" />
  <include href="http://<hostname>:
<port>/webservicess/rest/servicelocator/?XSD=getrestinterface.xsd"
xmlns="http://www.w3.org/2001/XMLSchema" />
  <include href="http://<hostname>:
<port>/webservicess/rest/servicelocator/?XSD=getrestinterfaces.xsd"
xmlns="http://www.w3.org/2001/XMLSchema" />
  <include href="http://<hostname>:
<port>/webservicess/rest/servicelocator/?XSD=removegrant.xsd" xmlns="
http://www.w3.org/2001/XMLSchema" />
</grammars><resources base="http://<hostname>:
<port>/webservicess/rest/servicelocator/">
  <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/servicelocator/">`

This information `http://<hostname>:`

`<port>/webservices/rest/servicelocator/` 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

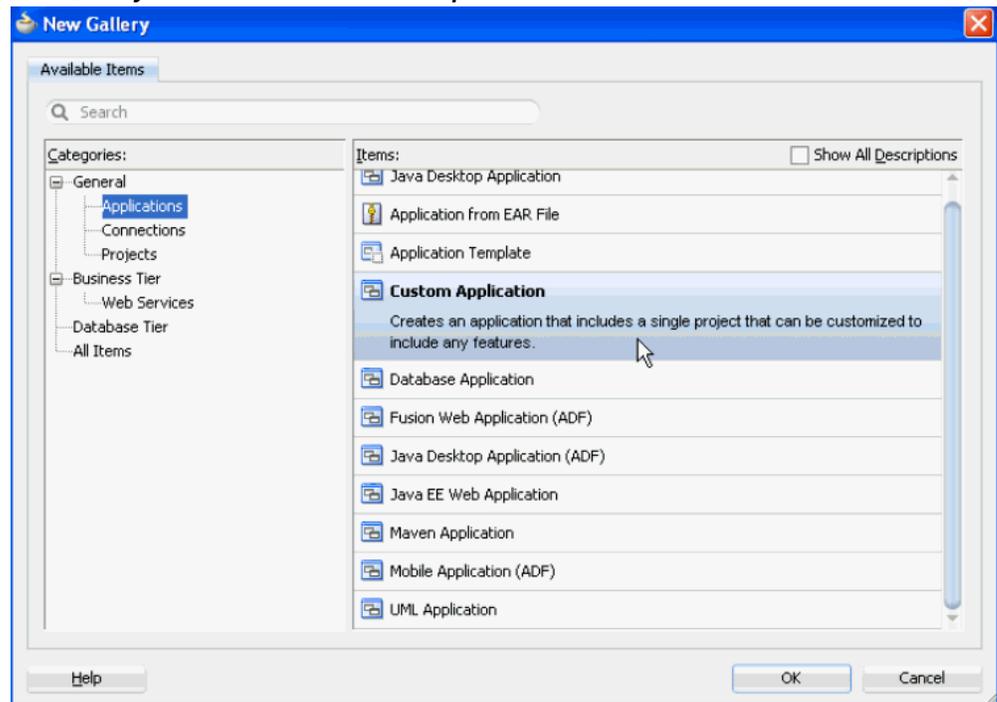
This section describes how to create a project with a Java class that will be used to invoke the REST Service Locator service.

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**.

New Gallery Window in Oracle JDeveloper



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/servicelocator";
    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 TLS-enabled environment. Additionally, you need to import the TLS certificate into your client JVM's keystore.

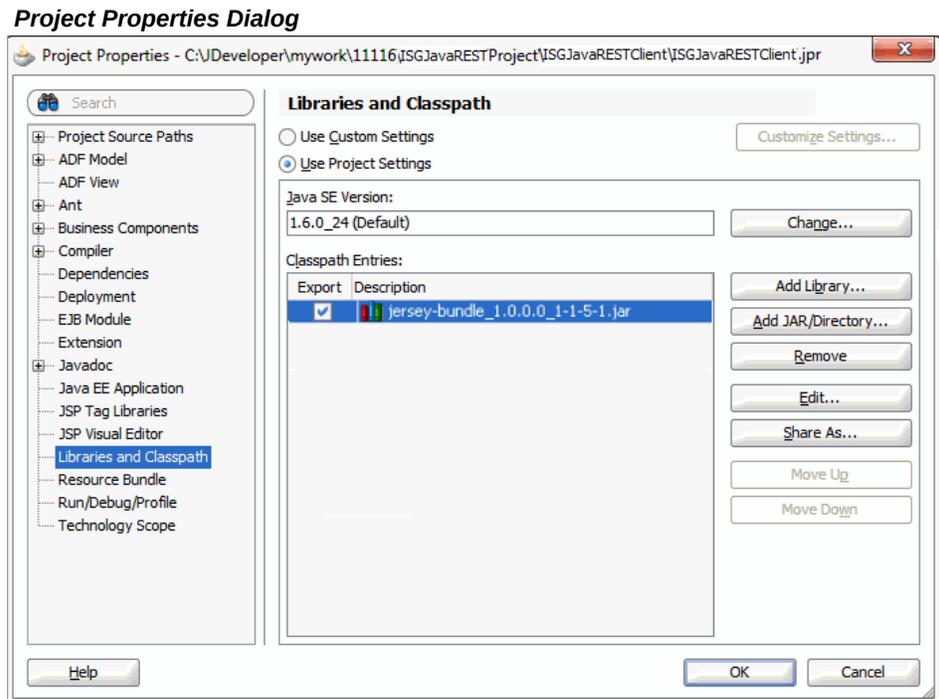
10. Replace `<EBS hostname>:<port>`, `<EBS username>`, and `<password>` with the actual values in the code.
11. Save your work by selecting **File > Save All**.
12. **Add required library for JSON format:**

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 run 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 and verify this process and check for successful compilation 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/servicelocator/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 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
  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" : "NotAnything",
      "serviceName" : "PLSQL:FND_PROFILE",
      "wadlUrl" : "http://<hostname>:<port>/webservices/rest/profile?WADL"
    },
    "ControlBean" : {
      "fields" : null,
      "filter" : null,
      "limit" : null,
      "offset" : null
    }
  }
}
```

Note: The message payload used here is an example. You should use the actual definition of the service in XSD and WADL.

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 runtime 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 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 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:

1. Creating and Compiling Custom Java APIs, page 4-17
2. Deploying Custom Java Classes and Source Files, page 4-34
3. Parsing and Uploading the Annotated Custom Java Bean Service to the Integration Repository, page 4-34
4. Deploying a Custom Java Bean Service, page 4-36
5. Creating a Security Grant, page 4-37
6. Recording Resource Information from Deployed WADL, page 4-38
7. Invoking a Custom REST Service from HTML Pages with Javascript, page 4-41

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 the 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 the 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 `ServiceException` 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. 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 'ISGJava_RESTDemo'.

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.

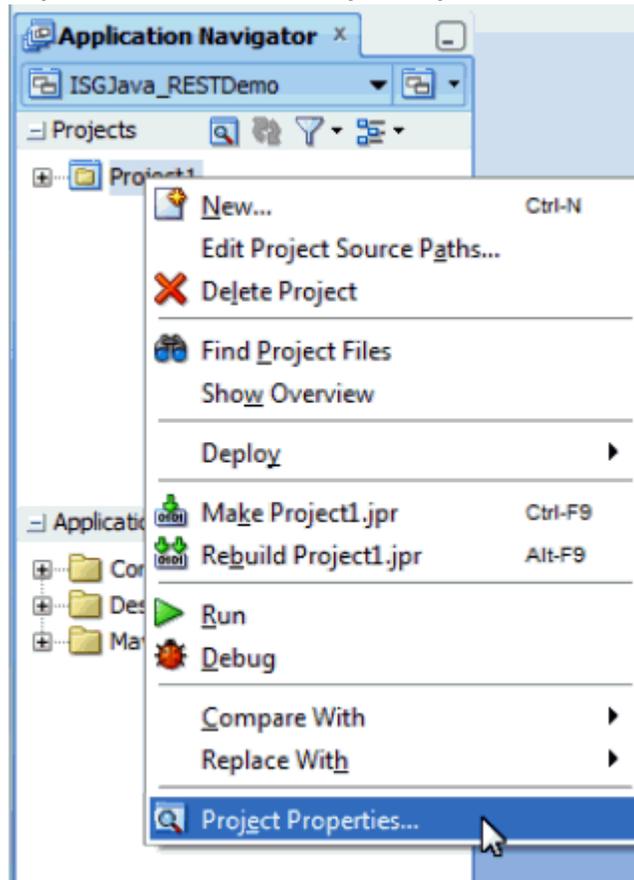
The newly created project should be visible in the Projects workspace.

6. **Add required libraries:**

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

1. Select and right-click on the project name "ISGJava_RESTDemo" you just created earlier to open a selection menu.
2. Select **Project Properties** from the menu.

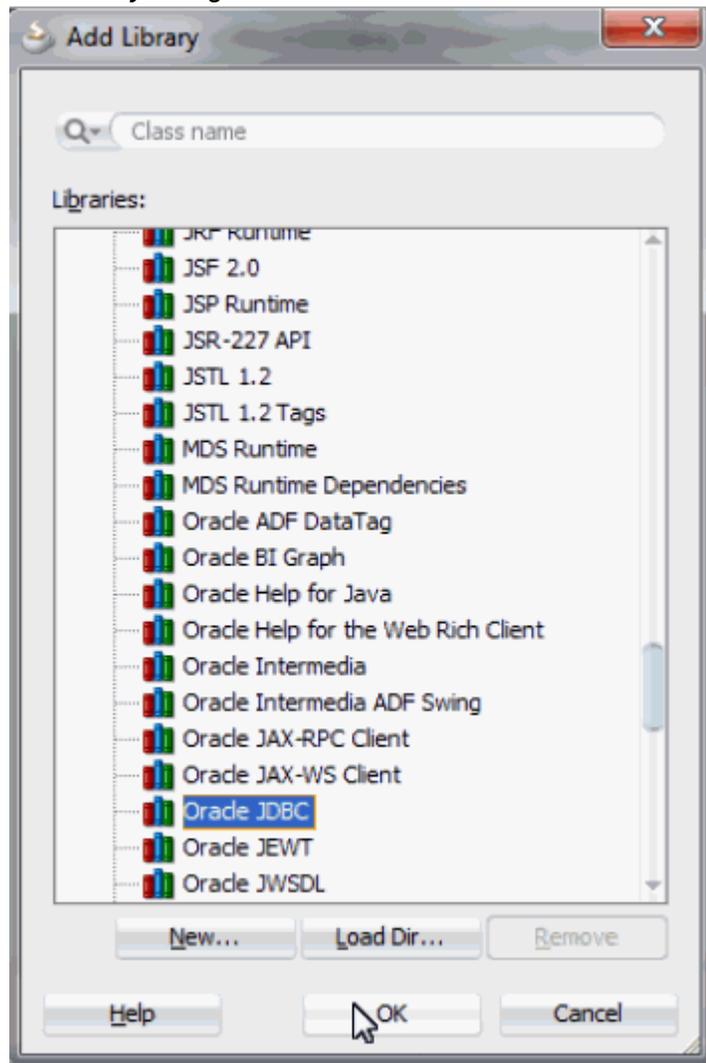
Drop-Down Menu to Select Project Properties



The Default Properties dialog box opens.

3. Select **Libraries and Classpath**, and click **Add Library**. The Add Library dialog box appears.
4. In the Add Library dialog box, select and expand the Extension folder and choose 'Oracle JDBC' from the list.

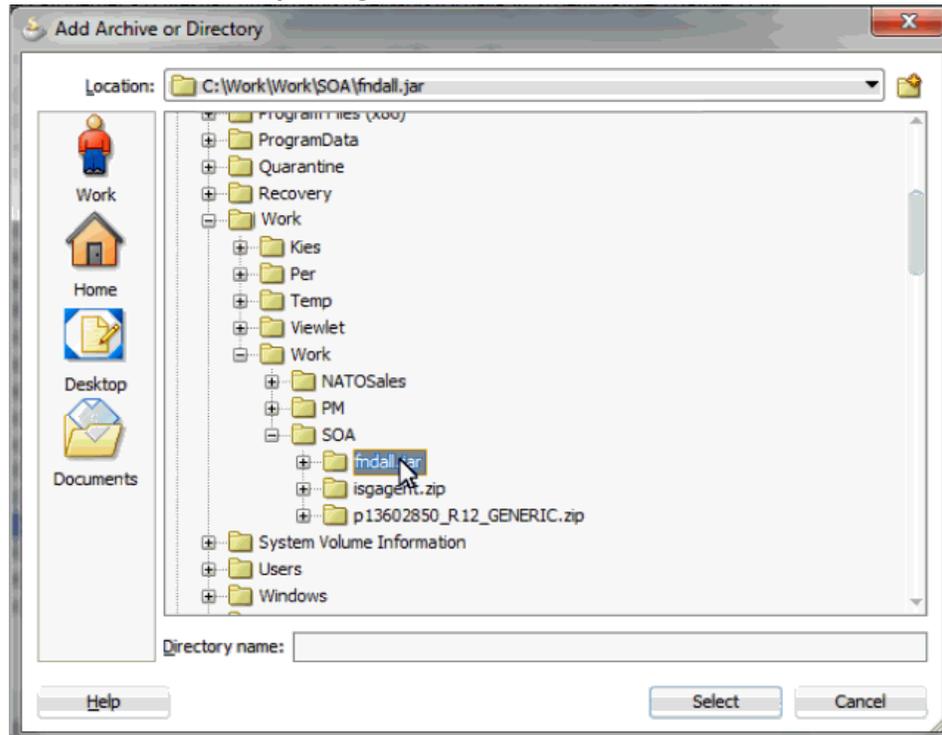
Add Library Dialog



Click **OK**. Oracle JDBC is added to the Classpath Entries section.

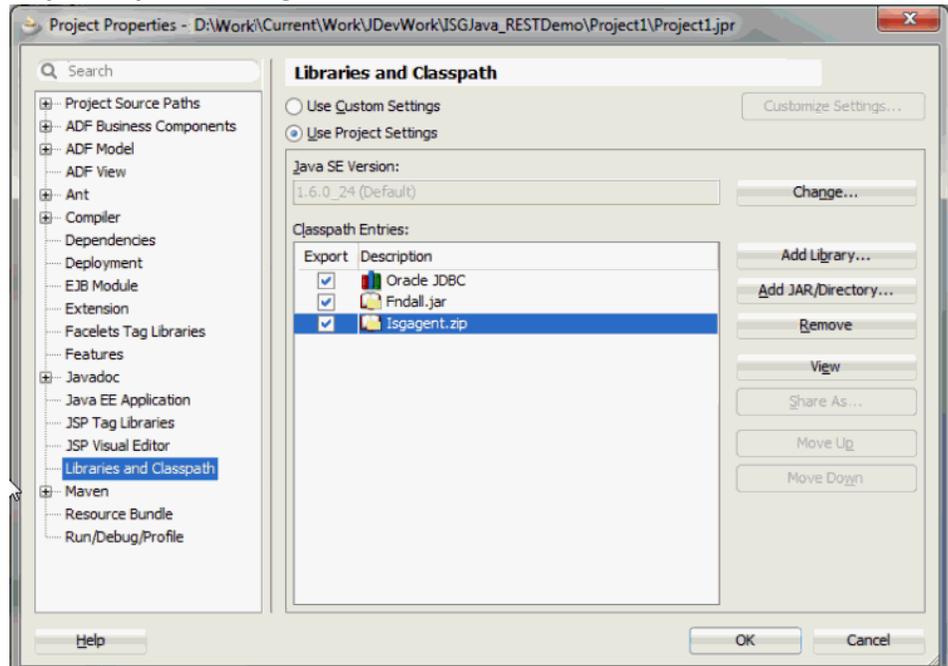
5. Click **Add JAR/Directory**. The Add Archive or Directory dialog box appears. Browse the directory and locate the 'fndall.jar' file. Click **Select** to add the selected 'fndall.jar' file.

Add Archive or Directory Dialog



Use the same approach to add 'isgagent.zip' file to the Classpath Entries region in the Project Properties dialog.

Project Properties Dialog with Added Libraries



Click **OK**. This project is now set up with the required libraries.

7. Select and right-click on the project name "ISGJava_RESTDemo" you just created and choose **New** from the drop-down selection menu.
8. In the New Gallery window, expand the General category and select 'Java'. In the Items list, select **Class**. Click **OK**.
9. In the Create Java Class dialog, create a Java class called `PersonBean.java` with the following information:
 - Name: `PersonBean`
 - Package: `oracle.apps.per.sample.beans`
 - Extends: `java.lang.Object`

Create Java Class Dialog

Enter the details of your new class.

Name:

Package:

Extends:

Optional Attributes

Implements:

Access Modifiers

public

package protected

Other Modifiers

<None>

abstract

final

Constructors from Superclass

Implement Abstract Methods

Main Method

Help OK Cancel

Click **OK**. The new class opens automatically in the source editor. Use the following Java code for PersonBean.java.

```

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;
}
}

```

10. Repeat steps 7, 8, and 9 to create the following two Java classes:

- `PerServiceException.java` with the following information:
 - Name: `PerServiceException`
 - Package: `oracle.apps.per.sample.common`
 - Extends: `java.lang.Object`

Use the following Java code for `PerServiceException.java`:

```
package oracle.apps.per.sample.common;

import oracle.apps.fnd.isg.app.ebs.rt.common.ServiceException;

public class PerServiceException extends ServiceException {

    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);
    }
}
```

- EmployeeInfo.java with the following information:
 - Name: EmployeeInfo
 - Package: oracle.apps.per.sample.service
 - Extends: java.lang.Object

Use the following Java code for EmployeeInfo.java:

```

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" +
\n" +
                                                "(SELECT PPF.PERSON_ID,
\n" +
                                                "PPF.FIRST_NAME,\n" +
\n" +
                                                "PPF.LAST_NAME,\n" +
\n" +
                                                "PPF.TITLE,\n" +
\n" +
                                                "PPF.FULL_NAME,\n" +
\n" +
                                                "PPF.SUFFIX,\n" +
\n" +
                                                "PPF.EMPLOYEE_NUMBER,
\n" +
                                                "PPF.EMAIL_ADDRESS,\n" +
+
                                                "PPF.WORK_TELEPHONE,\n" +
+
                                                "PAF.SUPERVISOR_ID,\n" +
+
                                                "PAF.
SUPERVISOR_ASSIGNMENT_ID,\n" +
\n" +
                                                "GREATEST(PPF.
EFFECTIVE_START_DATE, PAF.EFFECTIVE_START_DATE)
EFFECTIVE_START_DATE, \n" +
\n" +
                                                "LEAST(PPF.
EFFECTIVE_END_DATE, PAF.EFFECTIVE_END_DATE)
EFFECTIVE_END_DATE,\n" +
\n" +
                                                "FUR.USER_ID,\n" +
\n" +
                                                "FUR.USER_NAME\n" +
\n" +
                                                "FROM FND_USER FUR,
\n" +
\n" +
                                                "PER_ALL_ASSIGNMENTS_F
PAF\n" +
\n" +
                                                "WHERE PPF.PERSON_ID =
PAF.PERSON_ID (+)\n" +
\n" +
                                                "AND PPF.
BUSINESS_GROUP_ID = PAF.BUSINESS_GROUP_ID (+)\n" +
\n" +
                                                "AND (SYSDATE BETWEEN
PPF.EFFECTIVE_START_DATE\n" +
\n" +
                                                "AND PPF.
EFFECTIVE_END_DATE\n" +

```

```

"AND SYSDATE BETWEEN PAF.EFFECTIVE_START_DATE\n" +
"AND PAF.EFFECTIVE_END_DATE)\n" +
PPF.PERSON_ID\n" +
SUPERVISOR_ID = PRIOR PERSON_ID";
}
"AND FUR.EMPLOYEE_ID =
") E\n" +
"CONNECT BY

```

11. Save your work by selecting **File > Save All**.
12. Select and right-click on the project name "ISGJava_RESTDemo" and choose **Make Project1.jpr** from the drop-down selection menu. The compilation process starts. Verify that the compilation process is successful in the Message - Log window.

Deploying Custom Java Classes and Source Files

Once the custom Java API has been successfully created, the integration developer needs to copy these newly-created Java classes and source files to a target instance.

For information on how to deploy these custom Java classes, refer to Section 5: Deploying Custom Application Tier Objects, Deploying Customizations in Oracle E-Business Suite Release 12.2, My Oracle Support Knowledge Document 1577661.1.

After the successful deployment to a target instance, stop and restart the managed 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 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.

Verification of the Annotated Custom Interface Definition

```
-bash-3.2$ source /u01/R122_EBS/fs1/EBSapps/appl/... .env
-bash-3.2$ export JAVA_HOME=/u01/R122_EBS/fs1/EBSapps/comn/util/jdk32
-bash-3.2$ export CLASSPATH=/u01/R122_EBS/fs1/EBSapps/comn/java/lib/lsagent.zip:$CLASSPATH
-bash-3.2$ export CLASSPATH=/u01/R122_EBS/fs1/EBSapps/comn/clone/jlib/ojdbc6.jar:$CLASSPATH
-bash-3.2$ cd $JAVA_TOP
-bash-3.2$ cd oracle/apps/per/sample/service/
-bash-3.2$ ll
total 84
-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 1281 Jul 3 04:01 L7572911.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$
```

3. Run 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:
12.2=EmployeeInfo.java
```

Integration Repository Parser Command Process and Validation

```
-bash-3.2$ $IAS_ORACLE_HOME/perl/bin/perl $FND_TOP/bin/irep_parser.pl -g -v -username=sysadmin per:patch/115/java:EmployeeIn
fo.java:12.2=EmployeeInfo.java
# Interface Repository Annotation Processor, 12.0.0
#
# Generating annotation output.
# Processing file 'EmployeeInfo.java'.
# Using YAPP-based parser.
# Warning: Carriage return(s) seen in text.
# Java code is in package 'oracle.apps.per.sample.service'.
# Defining the class 'EmployeeInfo'.
# Found a class annotation for 'EmployeeInfo'.
# Found a member annotation for the method named getDirectReports
# Found a member annotation for the method named getAllReports
# Found a member annotation for the method named getPersonInfo
# Javadoc at line 158 with no IR annotation tags ignored.
# Done all files.
-bash-3.2$ 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 L7572911.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
```

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 username> @TWO_TASK 0 Y UPLOAD
$FND_TOP/patch/115/import/wfirep.lct EmployeeInfo_java.ildt
ORACLE Password:
```

5. Verify the generated log file to view the upload details.

Upload Details in the Log File

```
+-----+
Application Object Library: Version : 12.2
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/R122_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/R122_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 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**.

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' checkbox for the Get Person Details Java method. Leave the rest of preselected checkboxes unchanged.
 - In the REST Service Security region, ensure that at least one authentication type is selected.
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.

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 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 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>/webservicess/rest/empinfo/?
XSD=getallreports.xsd" xmlns="http://www.w3.org/2001/XMLSchema" />
  <include href="http://<hostname>:<port>/webservicess/rest/empinfo/?
XSD=getdirectreports.xsd" xmlns="http://www.w3.org/2001/XMLSchema" />
  <include href="http://<hostname>:<port>/webservicess/rest/empinfo/?
XSD=getpersoninfo.xsd" xmlns="http://www.w3.org/2001/XMLSchema" />
</grammars><resources base="http://<hostname>:
<port>/webservicess/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>
</resources>

```

```

</resource>
<resource path="/">
  <resource path="/">
    <param name="personId" style="template" required="true" type="xsd:int" />
  </resource>
  <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 an actual value (employee ID 13137 in this example) at runtime 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:
Replace `<hostname>`: `<port>`, `<EBS username>`, and `<password>` with the actual values in the code.

```

<!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,"<EBS username>", "<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 the path parameter **{personId}** to obtain the employee ID at runtime. The `getPersonInfo` service operation is then invoked through the HTTP GET request.

```

<resource path="/">
  <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

**Oracle E-Business Suite Integrated SOA Gateway - REST Services
Sample**

Path Parameter Example

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).

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.

Enter Employee ID	<input style="width: 80%;" type="text" value="13137"/>	<input type="button" value="Get Details"/>
Employee Name		
Manager Name		

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 the manager's name of

the employee.

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

Oracle E-Business Suite Integrated SOA Gateway - REST Services Sample

Path Parameter Example

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).

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.

Enter Employee ID	<input type="text" value="13137"/>	<input type="button" value="Get Details"/>
Employee Name	Taylor, Mr. Steve	
Manager Name	Bennett, Terrence G (Terry)	

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 the pagination control parameter **offset** defined in the Javascript to limit the number of records returned at runtime and for pagination when the `getDirectReports` service operation is invoked through the HTTP GET request.

```

<script language="javascript">
...
    var url = "http://<hostname>:
<port>/webservices/rest/empinfo/getDirectReports/?ctx_organid=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

**Oracle E-Business Suite Integrated SOA Gateway - REST Services
Sample**

Pagination Control Parameters Example

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.

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 user name and password.

After you entered the user name and password and clicked **OK**, the service returns the first three direct reports (employees) of the user you just entered in the table.

Click **Next** to display the next three records retrieved from the direct reports list.
Click **Previous** to display the last three records from the current direct reports list.

HTML Page with Direct Reports Shown as the Results

**Oracle E-Business Suite Integrated SOA Gateway - REST Services
Sample**

Pagination Control Parameters Example

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.

Employee Id	Employee Name
276	Watson, Mr. Phillip
283	Evans, George
281	Rhodes, Mr. Connor

Displaying 4 to 6 records.

Note: Employee details shown in the above screenshot 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.

Invoking an Application Module Service Using Token Based Authentication and XML Payload

REST Service Invocation Scenario

Take an Application Module Service called 'Approvals Management' (`oracle.apps.fnd.wf.worklist.service.rt.server.WFWorklistServiceAMImpl`) as an example to explain the REST service invocation. The **Get Open Notifications** (`getOpenNotifications`) REST service operation contained in this interface is used in this example to return a list of all open notifications for the current user.

When consequent HTTP requests are received, the `getOpenNotifications` service operation is invoked to fetch all open workflow notifications for the current user based on the input payload sent in XML format as a REST request. You can retrieve results in a batch by providing page size and page number in the request.

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 `getOpenNotifications` REST service operation can be invoked to return all open notification in XML format as part of the output response message.

Prerequisites to Use an Application Module Service

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

Obtaining Needed Library for This Example

To successfully invoke the REST service with payload in XML format, you need to obtain the following library available at `<$COMMON_TOP>/java/lib` directory for this example:

- `jersey-bundle_1.0.0.0_1-1-5-1.jar`

This library file 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, application context values can be passed in the 'RESTHeader' element before invoking a REST service.

These RESTHeader elements are *Responsibility*, *RespApplication*, *SecurityGroup*, *NLSLanguage*, and *Org_Id*.

For more information about the RESTHeader elements, see REST Header for Application Context, page 2-45.

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 4-49
2. Recording the Deployed WADL URL, page 4-51
3. Creating a Security, page 4-52
4. Creating a Project with a Java Class, page 4-53
5. Invoking a REST Service Using a Java Class, page 4-60

Deploying a REST Service

Use the following steps to deploy the Application Module Service called 'Approvals Management':

1. Log in to Oracle E-Business Suite as a user who has the Integration 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. 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: Application Module Services
4. Click **Go**.
Click the 'Approvals Management' interface name link to open the interface details page.
5. In the REST Web Service tab, enter the following information:

REST Web Service Tab

Overview **REST Web Service** Grants

* Service Alias Log Configuration Disabled [Configure](#)

REST Service Status Not Deployed

Service Operations

Expand All | Collapse All

Display Name	Internal Name	GET	POST	Grant
Approvals Management	oracle.apps.fnd.wf.worklist.service.rt.server.WFWorklistServiceAMImpl	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
Get Approval Types	getApprovalTypes	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
Get Open Notifications	getOpenNotifications	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
Get Pending Approvals By Approval Type	getPendingApprovalsByType	<input type="checkbox"/>	<input checked="" type="checkbox"/>	

REST Service Security

Personalize "REST Service Security"

*Authentication Type HTTP Basic Security Token

Tip: Use [Login Service](#) to obtain Security Token for given user credentials.

[Deploy](#)

- Service Alias: notification
The alias will be displayed as the service endpoint in the WADL and schema for the selected method or service operation which is `getOpenNotifications` in this example.
- In the Service Operations region, select 'Get Open Notifications' (`getOpenNotifications`).
The selected method will be exposed as a REST service operation.
- In the REST Service Security region, ensure that at least one authentication type is selected.

- 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 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 'Approvals Management' REST 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

Service Alias notification Log Configuration Disabled

REST Service Status Deployed [View WADL](#)

Service Operations

Display Name	Internal Name	GET	POST	Grant
Approvals Management	oracle.apps.fnd.wf.worklist.service.rt.server.WFWorklistServiceAMImpl			
Get Approval Types	getApprovalTypes		✓	
Get Open Notifications	getOpenNotifications		✓	
Get Pending Approvals By Approval Type	getPendingApprovalsByType		✓	

REST Service Security

*Authentication Type HTTP Basic Security Token

Tip: Use [Login Service](#) to obtain Security Token for given user credentials.

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

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

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

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

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

Creating a Security Grant

After deploying the 'Approvals Management' interface 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 Approvals Management service the administrator just deployed earlier.
3. In the interface details page of the selected Approvals Management service, click the Grants tab.
4. Select the `getOpenNotifications` method checkbox and then click **Create Grant**.
5. In the Create Grants page, select "All User" as the Grantee Type.

Note: In this example, the `getOpenNotifications` 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.

Interface Details Page with Grants Tab

Java Details : Approvals Management

Internal Name oracle.apps.fnd.wf.worklist.service.rt.server.WFWorklistServiceAMImpl Scope Public
Type Java Interface Source Oracle
Product Workflow Interface Subtype Application Module Services
Status Active
Business Entities [Mobile Optimized API](#) , [Workflow Notification](#)

Overview REST Web Service **Grants**

Select Object and	Create Grant	Revoke Grant			
Name ▲	Internal Name ▲	SOAP Service Operation ▲	REST Service Operation ▲	Grant ▲	
<input type="checkbox"/> Get Approval Types	getApprovalTypes		✓		
<input type="checkbox"/> Get Open Notifications	getOpenNotifications		✓		
<input type="checkbox"/> Get Pending Approvals By Approval Type	getPendingApprovalsByType		✓		

Creating a Project with a Java Class

This section describes how to create a project with a Java class (AMClient.java) and XML payload that will be used to invoke the 'Approvals Management' 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 'ISGAMClient'.

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 'AMClient'. 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.net.HttpURLConnection;
import java.net.URL;
import com.sun.jersey.core.util.Base64;
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 javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import org.w3c.dom.Document;
import org.w3c.dom.Element;

public class AMClient {
    public AMClient() {
        super();
    }

    private static final String xmlRequest1 = "<?xml version=\"1.0\"
encoding=\"utf-8\"?>\n" +
"<getOpenNotifications_Input>\n" +
"  <RESTHeader>\n" +
"    <NLSLanguage>AMERICAN</NLSLanguage>\n" +
"    <Org_Id>202</Org_Id>\n" +
"    <RespApplication>SYSADMIN</RespApplication>\n" +
"    <Responsibility>SYSTEM_ADMINISTRATOR</Responsibility>\n" +
"    <SecurityGroup>STANDARD</SecurityGroup>\n" +
"  </RESTHeader>\n" +
"<InputParameters>\n" +
"  <pageSize>3</pageSize>\n" +
"  <pageNumber>1</pageNumber>\n" +
"</InputParameters>\n" +
"</getOpenNotifications_Input>";

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

        String loginUrl = baseUrl + "/login";
        URL url = new URL(loginUrl);
        //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("Accept", "application/xml");
        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
DocumentBuilderFactory factory = DocumentBuilderFactory.
newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
ByteArrayInputStream input = new ByteArrayInputStream
(response.getBytes("UTF-8"));
Document doc = builder.parse(input);
Element eAccessToken = (Element) doc.
getElementsByTagName("accessToken").item(0);
String strAccessToken = eAccessToken.getTextContent();
Element eTokenName = (Element) doc.getElementsByTagName
("accessTokenName").item(0);
String strTokenName = eTokenName.getTextContent();

return (new String[] { strTokenName, strAccessToken });
}

/**
 * This method reads response sent by the server and returns it
in a string representation.
 */
private static String readHttpResponse(URLConnection 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());
}

/**

```

```

* This method invokes Workflow Notification rest service using the
accessTokenName and accessToken values returned by EBS login service
*/
public static void invokeRESTService(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/xml");
    //Adding the accessTokenName and accessToken as Cookies
    conn.addRequestProperty("Cookie", tokenName + "=" +
tokenValue);
    conn.setRequestProperty("Accept", "application/xml");
    conn.setUseCaches(false);
    conn.setDoInput(true);
    conn.setDoOutput(true);
    //Send request
    OutputStreamWriter wr = new OutputStreamWriter(conn.
getOutputStream());
    wr.write(xmlRequest1.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);
}

public static void main(String[] args) throws Exception {
    AMClient amClient = new AMClient();
    String baseUrl = "http://<EBS hostname>:
<port>/webservices/rest";
    String svcUrlStr1 = baseUrl + "/"
notification/getOpenNotifications/";
    System.out.println("Calling EBS Login Service");
    //Get Access Token by invoking AOL Login Service
    String[] token = getAccessToken(baseUrl, "<EBS username>", "
<password>");
    System.out.println("Obtained EBS Access Token");
    //Invoke REST service using the Access Token
    invokeRESTService(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 TLS-enabled

environment. Additionally, you need to import the TLS certificate into your client JVM's keystore.

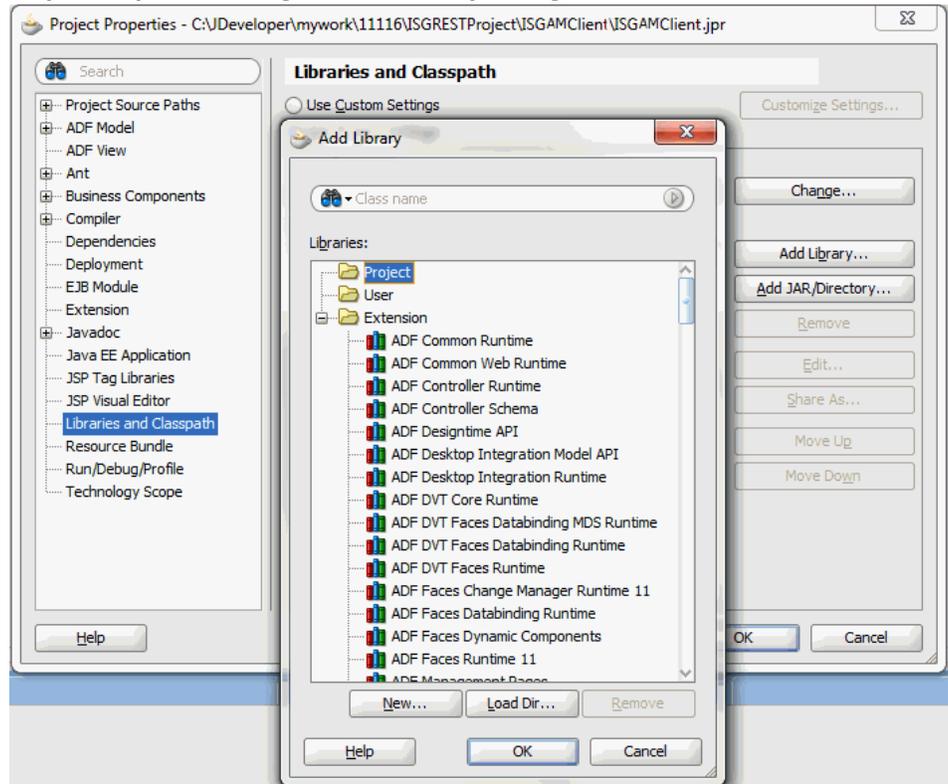
9. Replace `<EBS_hostname>:<port>`, `<EBS_username>`, and `<password>` with the actual values in the code.

10. Add the required library to process XML payload:

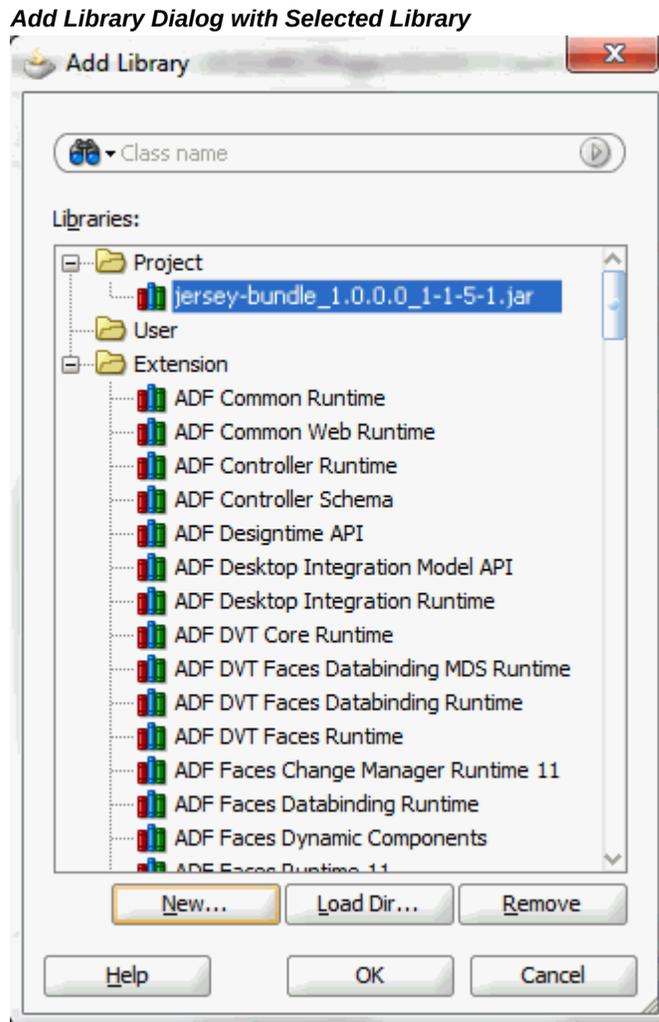
Use the following steps to add the required library file to the project properties.

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.

Project Properties Dialog and Add Library Dialog



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.

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 `AMClient.java`, you need to compile and run the process to invoke the 'Approvals Management' REST service.

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

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

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

Monitor and verify this process and check for successful compilation in the Log window.

Viewing Output Message

When the 'Approvals Management' REST service is successfully invoked, the following response appears:

```

Calling EBS Login Service
Obtained EBS Access Token
Response code - 200
Response is :
<?xml version = '1.0' encoding = 'UTF-8'?>
<getOpenNotifications_Output xmlns:ns="http://xmlns.oracle.
com/apps/fnd/rest/output">
  <ns:OutputParameters>
    <ns:Output>
      <ns:WFAApprovalsBean
xmlns:ns2="http://xmlns.oracle.
com/apps/fnd/wf/worklist/service/rt/artifacts/fault/"
xmlns:ns1="http://xmlns.oracle.
com/apps/fnd/wf/worklist/service/rt/artifacts/notificationdetails/"
xmlns:ns="http://xmlns.oracle.
com/apps/fnd/rest/notification/getOpenNotifications"
xmlns:ns0="http://xmlns.oracle.
com/apps/fnd/wf/worklist/service/rt/artifacts/directoryservices/">
        <ns:output>
          <notifications count="23">
            <notification>
              <NotificationId>7857291</NotificationId>
              <Subject>Standard Purchase Order 6562 has been
approved</Subject>
              <Language>US</Language>
              <BeginDate>30-Sep-2010 01:25:03</BeginDate>
              <Status>Open</Status>
              <FromUser>Brown, Casey</FromUser>
              <Name>POAPPRV</Name>
              <SentTime>30-Sep-2010</SentTime>

<NotificationLink>OA.jsp?OAFunc=FND_WFNTF_DETAILS&NtfId=7857291&...
</NotificationLink>

            <ns1:result/>
          </notification>
          <notification>
            <NotificationId>7857248</NotificationId>
            <Subject>Standard Purchase Order 6561 has been
approved</Subject>
            <Language>US</Language>
            <BeginDate>30-Sep-2010 01:20:27</BeginDate>
            <Status>Open</Status>
            <FromUser>Brown, Casey</FromUser>
            <Name>POAPPRV</Name>
            <SentTime>30-Sep-2010</SentTime>

<NotificationLink>OA.jsp?
OAFunc=FND_WFNTF_DETAILS&NtfId=7857248&...</NotificationLink>

            <ns1:result/>
          </notification>
          <notification>
            <NotificationId>7747986</NotificationId>
            <Subject>Standard Purchase Order 6549 has been
approved</Subject>
            <Language>US</Language>
            <BeginDate>13-Sep-2010 17:00:54</BeginDate>
            <Status>Open</Status>
            <FromUser>Stock, Pat</FromUser>
            <Name>POAPPRV</Name>
            <SentTime>13-Sep-2010</SentTime>

<NotificationLink>OA.jsp?
OAFunc=FND_WFNTF_DETAILS&NtfId=7747986&...</NotificationLink>

```

```
<ns1:result/>
  </notification>
  </notifications>
</ns:output>
</ns:WFAApprovalsBean>
</ns:Output>
<ns:ControlBean>
  <ns:filter/>
  <ns:limit/>
  <ns:offset/>
  <ns:sort/>
</ns:ControlBean>
</ns:OutputParameters>
</getOpenNotifications_Output>
```

In this example, the response message in XML format provides three open workflow notifications in the first page.

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 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 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 represented in WSDL can be used in creating a SOA composite application with 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. This can be done by dequeuing the message to retrieve outbound data from Oracle E-Business Suite. 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 discussed 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 a SOA Composite Application with BPEL Process at Runtime, page 5-22
- Using XML Gateway Outbound Through Subscription Model, page 5-30
 - Using XML Gateway Outbound Messages in Creating a SOA Composite Application with BPEL Process at Design Time, page 5-30
 - Deploying and Testing a SOA Composite Application with BPEL Process at Runtime, page 5-25

For the examples described in the following sections, Oracle JDeveloper 11g (11.1.1.6.0) is used as a design-time tool to create a SOA composite application with BPEL process and Oracle SOA Suite 11g (11.1.1.6.0) is used for the process deployment.

Note: While using Oracle JDeveloper with other Oracle Fusion Middleware components (such as Oracle SOA Suite), to enable SOA technologies, you need to manually download Oracle SOA Suite Composite Editor, a JDeveloper's extension for SOA technologies. For more information on installing additional Oracle Fusion Middleware design time components, see the *Oracle Fusion Middleware Installation Guide for Oracle JDeveloper*.

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 SOA Composite with BPEL Process at Runtime, page 5-22

Using XML Gateway Inbound Services at Design Time

SOA Composite Application with BPEL Process Scenario

Consider the XML Gateway Inbound Process PO XML Transaction as an example to explain the SOA Composite application with 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 the ECX_INBOUND queue. Agent Listeners running on the ECX_INBOUND queue would enable further processing by the Execution Engine. Oracle XML Gateway picks up this XML message, does trading partner validation, and inserts order data to Order Management Application.

When the SOA Composite application with BPEL process has been successfully invoked after deployment, the same order information will be inserted into the Order Management table once a purchase order is created.

Prerequisites to Configure a SOA Composite Application with BPEL Process Using an XML Gateway Inbound Service

Before performing the design-time tasks for XML Gateway Inbound services, ensure the following tasks are in place:

- An integration administrator or an integration developer needs to generate a web service first. The administrator will then deploy the generated service to an Oracle SOA Suite WebLogic managed 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.
- A trading partner should be available for receiving the XML documents.
- ECX Inbound Agent Listener and ECX Transaction Agent Listener must be up and running.

Deploying an XML Gateway Inbound Web Service Composite

Once a web service for the selected XML Gateway inbound map has been successfully created, the integration administrator must deploy the service from Oracle Integration Repository to an Oracle SOA Suite WebLogic managed server.

For example, the following steps must be performed first before the integration developer can create a SOA Composite application with BPEL process by using the deployed WSDL:

1. To generate a web service, the integration administrator or the integration

developer locates the interface definition first (such as an XML Gateway inbound interface `INBOUND:Process Purchase Order XML Transaction (ONT:POI)`) and clicks **Generate** in the interface details page.

Once the service has been successfully generated, the Web Service Status field changed from 'Not Generated' to 'Generated' in the Web Service region.

For detailed instructions 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, the integration administrator selects one authentication type before clicking **Deploy**. The deployed service in Oracle SOA Suite is an active service and is ready to accept new SOAP requests.

Once the service has been successfully deployed, the selected authentication type will be displayed along with 'Deployed' with 'Active' state in the Web Service Status field. For more information on securing web services with the authentication type, 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 and Undeploying SOAP Web Services, Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

Searching and Recording a WSDL URL

The integration developer needs 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.

For information on how to search for an interface and review the interface details, see *Searching and Viewing Integration Interfaces, page 2-1*.

Populating XML Gateway Header Variables

Certain variables in the BPEL PM must be populated in order to provide XML Gateway header information for Oracle E-Business Suite. `MESSAGE_TYPE`, `MESSAGE_STANDARD`, `TRANSACTION_TYPE`, `TRANSACTION_SUBTYPE`, `DOCUMENT_NUMBER` and `PARTY_SITE_ID` are the mandatory header variables that need to be populated in order for the XML transaction to complete successfully.

Refer to *Adding an Assign Activity, page 5-17*.

Setting Up a Trading Partner for Receiving XML Documents

Use the following steps to set up a trading partner:

1. Log in to Oracle E-Business Suite as a user who has the XML Gateway responsibility.
2. Click the **Define Trading Partners** link from the Navigator.

3. The Trading Partner Setup form is displayed. Search an existing trading partner by selecting **View > Query by Example > Enter** and enter the following information:
 - Trading Partner Type: Customer
 - Trading Partner Name: Example Inc.
 - Trading Partner Site: 401 Island Parkway Redwood Shores, CA 94065

Run the query by selecting **View > Query by Example > Enter** to locate the trading partner "Example Inc." along with existing transactions.

For information on setting up a trading partner, see Trading Partner Setup, *Oracle XML Gateway User's Guide*.

Trading Partner Setup Form

The screenshot shows the 'Trading Partner Setup' window. At the top, there are input fields for 'Operating Unit' (Vision Operations), 'Trading Partner Type' (Customer), 'Trading Partner Name', 'Trading Partner Site', and 'Company Admin Email'. Below these are 'User Setup' and 'Code Conversion' buttons. The main section is 'Trading Partner Details', which contains a table with the following data:

Transaction Type	Transaction SubType	Standard Code	External Transaction Type	External Transaction SubType	Direction	Map	Connection/Hub	Protocol Type
ONT	POA	OAG	PO	ACKNOWLEDG	OUT	ONT_3A4A_O	DIRECT	SMTP
ONT	CPO	OAG	PO	CANCEL	IN	ONT_3A9R_O		
ONT	POI	OAG	PO	PROCESS	IN	ONT_3A4R_O		
ONT	SSO	OAG	SALESORDE	SHOW	OUT	PROCESS_P	DIRECT	SMTP

Enter the following trading partner details:

- Transaction Type: ONT
- Transaction SubType: POI
- Standard Code: OAG
- External Transaction Type: PO

- External Transaction SubType: PROCESS
- Direction: IN
- Map: ONT_3A4R_OAG72_IN
- Source Trading Partner Location Code: Example-01

4. Save your entry.

Ensuring Agent Listeners Are Up and Running

Make sure that agent listeners on the ECX_INBOUND and 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 as a user who has 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 listeners, and select Start from the **Actions** box. Click **Go** if they are not up and running.

SOA Composite Application with BPEL Process Creation Flow

Based on the XML Gateway Inbound Process PO XML Transaction business scenario, the following design-time tasks are included in this chapter:

1. Create a New SOA Composite Application with BPEL Process, page 5-7
Use this step to create a new SOA Composite application with BPEL process 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 to the request application.
2. Create a Partner Link, page 5-11
Use this step to create a partner link to allow the inbound message to be inserted to Oracle E-Business Suite.
3. Add a Partner Link for File Adapter, page 5-12
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. Get the XML message details that is received from the Receive activity.
2. Enqueue the purchase order information to the ECX_INBOUND queue.

5. Add Assign Activities, page 5-17

Use this step to create two Assign activities in order to:

1. Pass XML message obtained from the first Invoke activity to the second Invoke activity.
2. Pass ECX header variables to the second Invoke activity as input variables.

For general information and how to create SOA composite applications using BPEL process service component, see the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite* for details.

Creating a New SOA Composite Application with BPEL Process

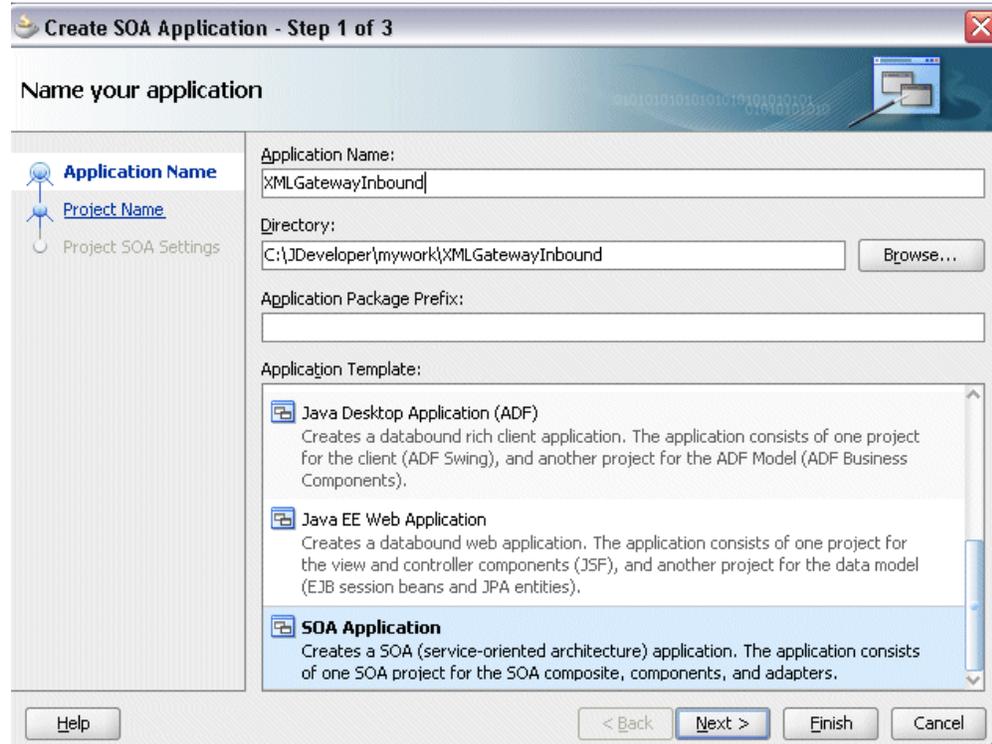
Use this step to create a new SOA composite application that will contain various BPEL process activities.

To create a new SOA composite application with BPEL process:

1. Open Oracle JDeveloper.
2. Click **New Application** in the Application Navigator.

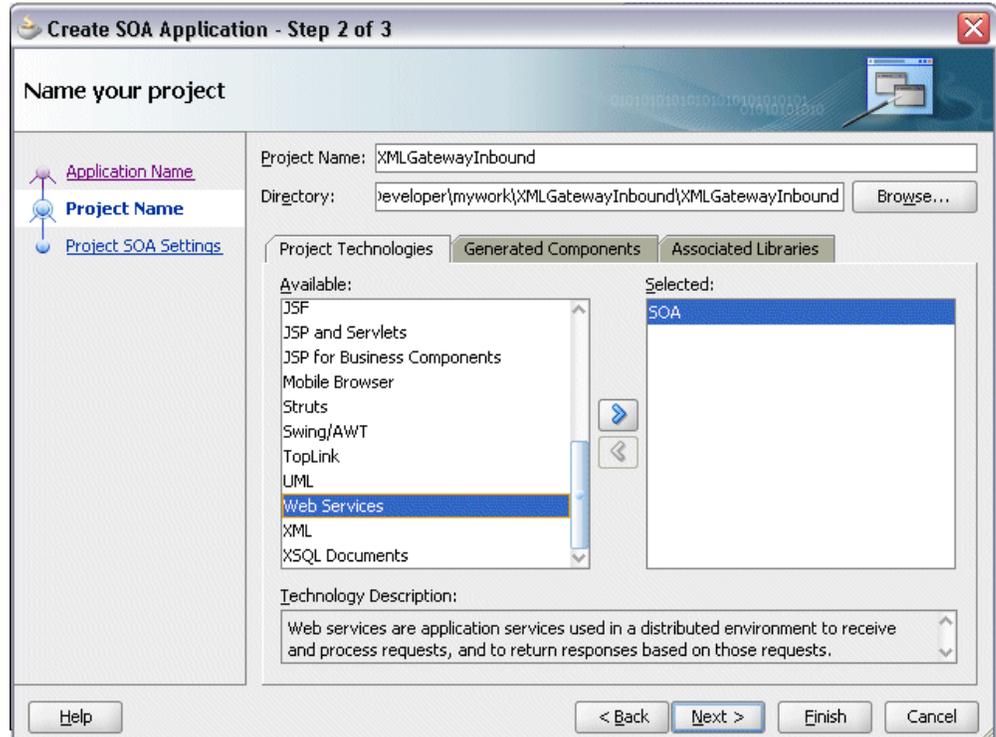
The "Create SOA Application - Name your application" page is displayed.

The Create SOA Application - Name your application Page



3. Enter an appropriate name for the application in the Application Name field and select **SOA Application** from the Application Template list.
Click **Next**. The "Create SOA Application - Name your project" page is displayed.

The Create SOA Application - Name your project Page



4. Enter an appropriate name for the project in the Project Name field, for example, XMLGatewayInbound.
5. In the Project Technologies tab, select 'Web Services' and ensure that **SOA** is selected from the Available technology list to the Selected technology list.
Click **Next**. The "Create SOA Application - Configure SOA settings" page is displayed.
6. Select **Composite With BPEL Process** from the Composite Template list, and then click **Finish**. You have created a new application, and a SOA project. This automatically creates a SOA composite.
The Create BPEL Process page is displayed.

The Create BPEL Process Page

BPEL Process

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

BPEL 1.1 Specification BPEL 2.0 Specification

Name: XMLGInbound

Namespace: http://xmlns.oracle.com/XMLGatewayInbound1/XMLGatewayInbound/XMLGInbound

Template: Synchronous BPEL Process

Service Name: xmlginbound_client

Expose as a SOAP service

Transaction: required

Input: xmlns.oracle.com/XMLGatewayInbound1/XMLGatewayInbound/XMLGInbound}process

Output: cle.com/XMLGatewayInbound1/XMLGatewayInbound/XMLGInbound}processResponse

Help OK Cancel

7. Leave the default **BPEL 1.1 Specification** selection unchanged. This creates a BPEL project that supports the BPEL 1.1 specification.

Enter an appropriate name for the BPEL process in the Name field, for example, XMLGInbound.

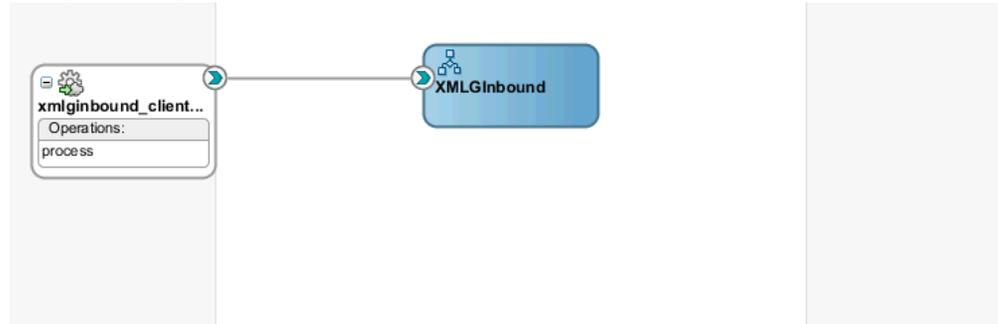
Select **Synchronous BPEL Process** in the Template field.

Select **required** from the Transaction drop-down list. Click **OK**.

A synchronous BPEL process is created with the Receive and Reply activities. The required source files including bpe1 and wsdl, using the name you specified (for example, XMLGInbound.bpe1 and XMLGInbound.wsdl) and composite.xml are also generated.

8. Navigate to SOA Content > Business Rules and click the composite.xml to view the composite diagram.

Composite Diagram



Double click on the XMLGIInbound component to open the BPEL process.

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 Oracle JDeveloper, place your mouse in the right side of the Partner Links area and right click to select **Create Partner Link...** from the pull-down menu. Alternatively, you can drag and drop **Partner Link** from the **BPEL Constructs** list into the right Partner Link swim lane of the process diagram.

The Create Partner Link window appears.

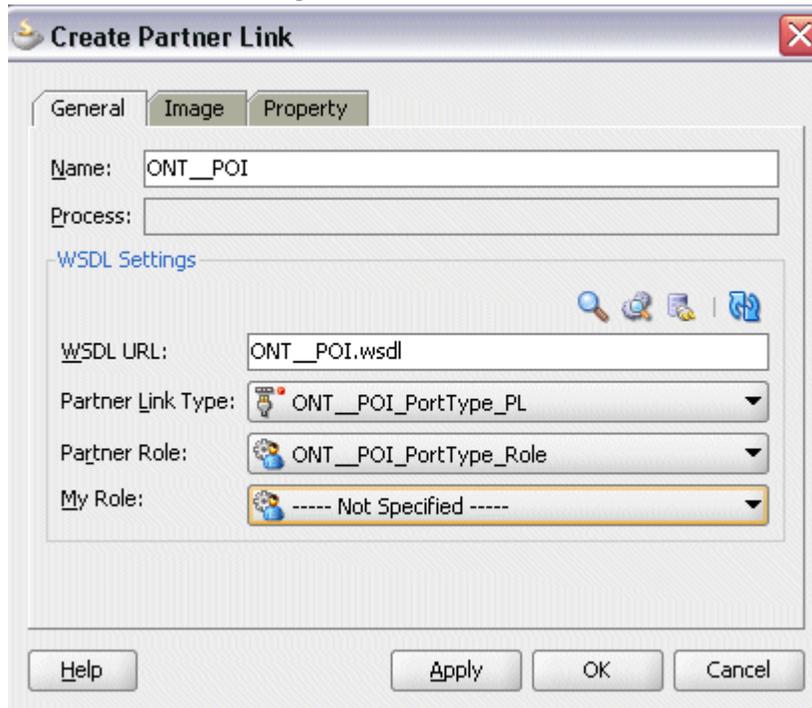
2. Copy the WSDL URL corresponding to the XML Gateway inbound map `INBOUND:Process Purchase Order XML Transaction (ONT:POI)` that you recorded earlier from the Integration Repository and paste it in the WSDL File field.

Press the **[Tab]** key.

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. You can manually enter the partner link name as ONT__POI.

Create Partner Link Dialog



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

Click **Apply** and **OK**.

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.

Partner Link is added to the Partner Links section in the BPEL process diagram.

Double click the `ONT__POIWrapper.wsdl` and click the Source tab to view the source WSDL description for the partner link you just created. Replace the reference WSDL with the deployed service end point WSDL. Save your changes by selecting **File > Save All**.

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 Oracle JDeveloper, drag and drop the **File Adapter** service from the **BPEL Services** into the right Partner Link swim lane 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**. The Adapter Interface dialog box appears.
5. Select the **Define from operation and schema (specified later)** radio button and click **Next**. The Operation dialog box appears.

Operation Dialog

Operation

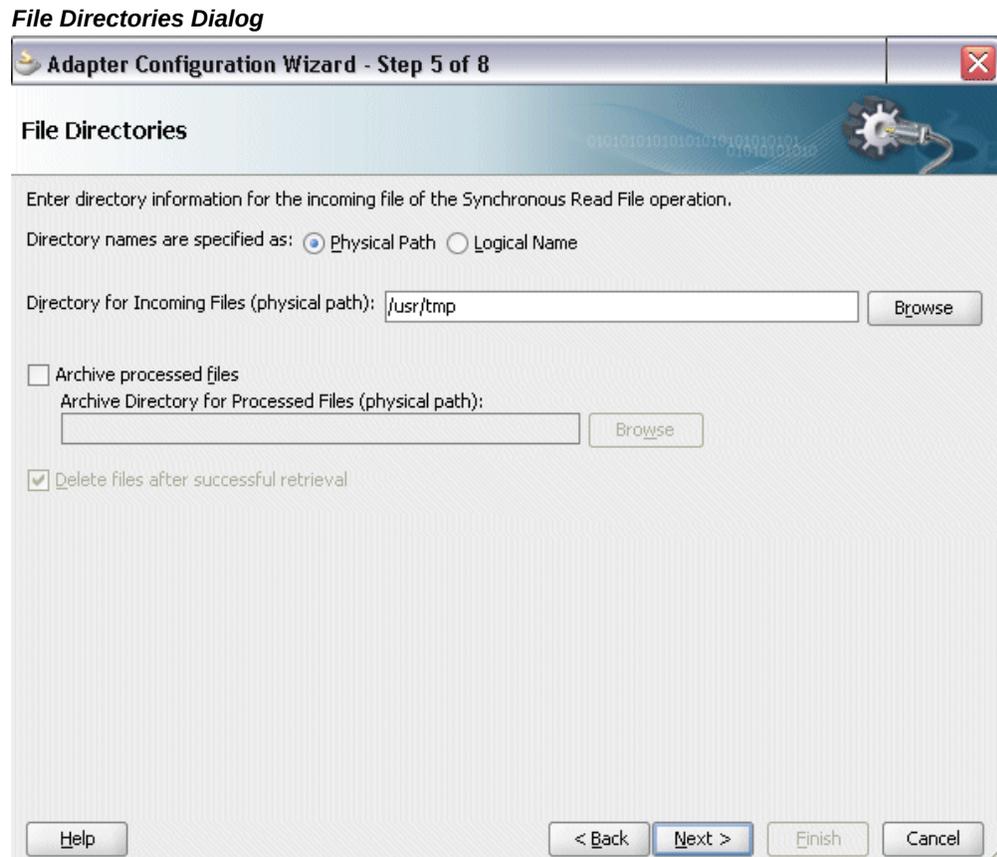
The File Adapter supports four operations. There is a Read File operation that polls for incoming files in your local file system, a Write File operation that creates outgoing files, a Synchronous Read File operation that reads the current contents of a file, and a List Files operation that lists file names in specified locations. Specify the Operation type and Operation Name. Only one operation per Adapter Service may be defined using this wizard.

Operation Type: Read File
 Write File
 Synchronous Read File
 List Files

Operation Name:

Help < Back Next > Finish Cancel

6. 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.



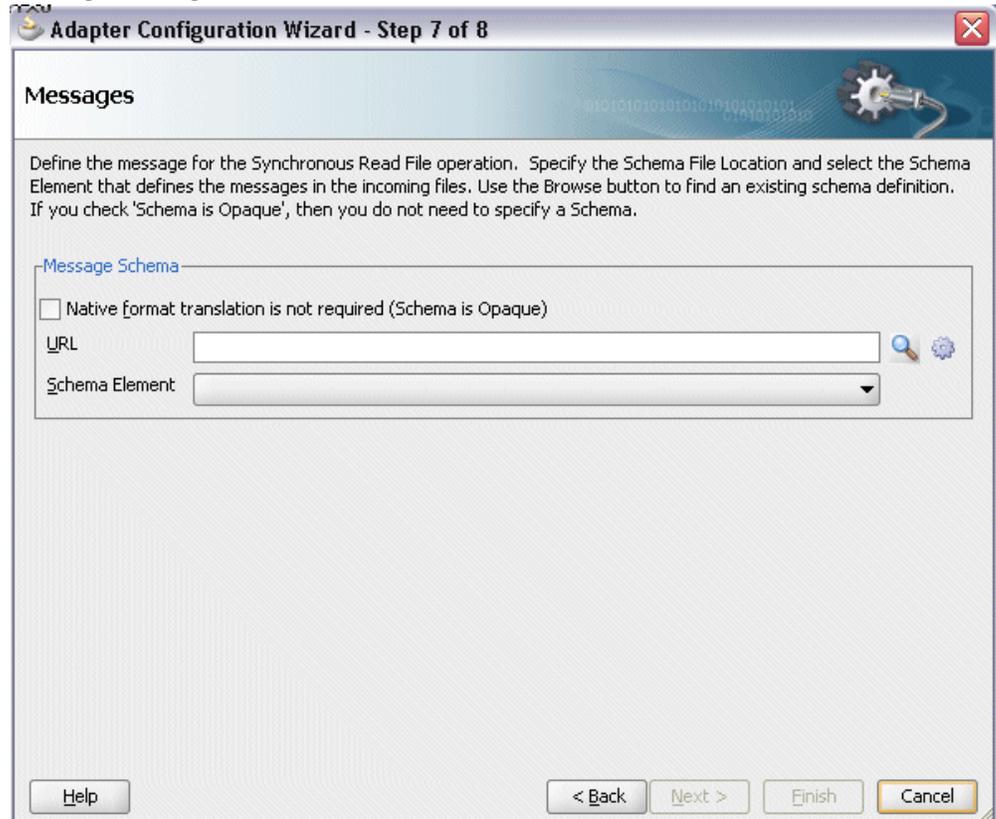
7. Select the **Physical Path** radio button and enter the input payload file directory information. For example, enter `/usr/tmp/`.

Note: To be able to locate the file from the physical directory that you specified here, you must first place the input payload file (such as `order_data_xmlg.xml`) to the specified directory.

Click **Next** to open the File Name dialog box.

8. 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.

Messages Dialog



9. Select **Browse for schema file** in front of the URL field.

The Type Chooser window is displayed.

Click the **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:`

```
//<soa_suite_hostname>:<port>/soa-  
infra/services/default/<jndi_name>_XMLGATEWAY_ONT__POI/ONT__PO  
I_Service?XSD=xsd/PROCESS_PO_007.xsd
```

Schema location for your service can be found from the service WSDL URL (for example, `http://<soa_suite_hostname>:<port>/soa-infra/services/default/<jndi_name>_XMLGATEWAY_ONT__POI/ONT__POI_Service?wsdl`).

Select the **Copy to Project** checkbox and click **OK**.

10. The Localize Files window appears. Ensure the **Maintain original directory structure for imported files** checkbox is selected and click **OK**.

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

11. Expand the imported schema folder and select PROCESS_PO_007 from the PROCESS_PO_007.xsd. Click **OK**. The selected PROCESS_PO_007.xsd is displayed as URL and the PROCESS_PO_007 element is selected as Schema Element.

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.

Under applications window, navigate to file GetXMLMsg_file.jca. Set value of property "DeleteFile" to "false".

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 Oracle JDeveloper, expand the **BPEL Constructs** from the Component Palette. Drag and drop the **Invoke** activity into the center swim lane of the process diagram, between the **receiveInput** and **replyOutput** activities.
2. Link the **Invoke** activity to the GetXMLMsg service. The Edit Invoke dialog box appears.
3. Enter a name for the **Invoke** activity (such as 'Invoke_Msg') 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 an input variable name. You can also accept the default name. Select **Global Variable** and click **OK**.
5. Click the **Create** icon next to the **Output Variable** field to create a new variable. The Create Variable dialog box appears.
6. Enter an input variable name. You can also accept the default name. Select **Global Variable** and click **OK**.
7. 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 Oracle JDeveloper, expand the **BPEL Constructs** from the Component Palette. Drag and drop the second **Invoke** activity into the center swim lane of the process diagram, after the first **Invoke** activity and the **replayOutput** activity.
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. Enter an input variable name. You can also accept the default name. Select **Global Variable** and click **OK**.
5. Click **Apply** and then **OK** in the Edit Invoke dialog box to finish configuring the Invoke activity.
6. The process diagram appears.

Adding Assign Activities

This step is to configure two Assign activities:

1. To pass XML message as an input to the Invoke activity for enqueueing 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 Oracle JDeveloper, expand the **BPEL Constructs** from the Component Palette. Drag and drop the **Assign** activity into the center swim lane of the process diagram between the two **Invoke** activities you just created earlier.
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. Select the Copy Rules tab and expand the source and target trees:
 - In the From navigation tree, navigate to **Variables > Process > Variables > Invoke_Msg_SynchRead_OutputVariable** and select **Process_PO_007**.

- In the To navigation tree, navigate to **Variables > Process > Variables > Invoke_ONT__POI_InputVariable** and select **body**.

Drag the source node (Process_PO_007) to connect to the target node (body) that you just identified. This creates a line that connects the source and target nodes. The copy rule is displayed in the From and To sections at the bottom of the Edit Assign dialog box.

5. 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. In Oracle JDeveloper, expand the **BPEL Constructs** from the Component Palette. Drag and drop the **Assign** activity into the center swim lane of the process diagram between the first Assign and the second **Invoke** activities you just created earlier.

Add the second Assign activity by dragging and dropping the **Assign** activity into the center swim lane of the process diagram between the SetOrderXML **Assign** activity and the second **Invoke** activity.

2. Repeat Step 2 to Step 3 described in creating the first Assign activity to add the second Assign activity called 'SetECXHeader'.

3. Select the Copy Rules tab and expand the source and target trees:

- Click the Expression icon to invoke the Expression Builder dialog.

Enter 'XML' in the Expression box. Click **OK**. The Expression icon with the expression value ('XML') appears in the center of the Edit Assign dialog, between the From and To navigation tree nodes.

- In the To navigation tree, navigate to **Variables > Process > Variables > Invoke_ONT__POI_InputVariable > header > ns1:SOAHeader > ns4:ECXMSG** and select **MESSAGE_TYPE**.

The XPath field should contain your selected entry.

Drag the Expression icon to connect to the target node (MESSAGE_TYPE) that you just identified. This creates a line that connects the source and target nodes. The copy rule is displayed in the From and To sections at the bottom of the Edit Assign dialog box.

4. Use the same mechanism described in step 3 to assign values to the following parameters:

- MESSAGE_STANDARD: 'OAG'
- TRANSACTION_TYPE: 'PO'

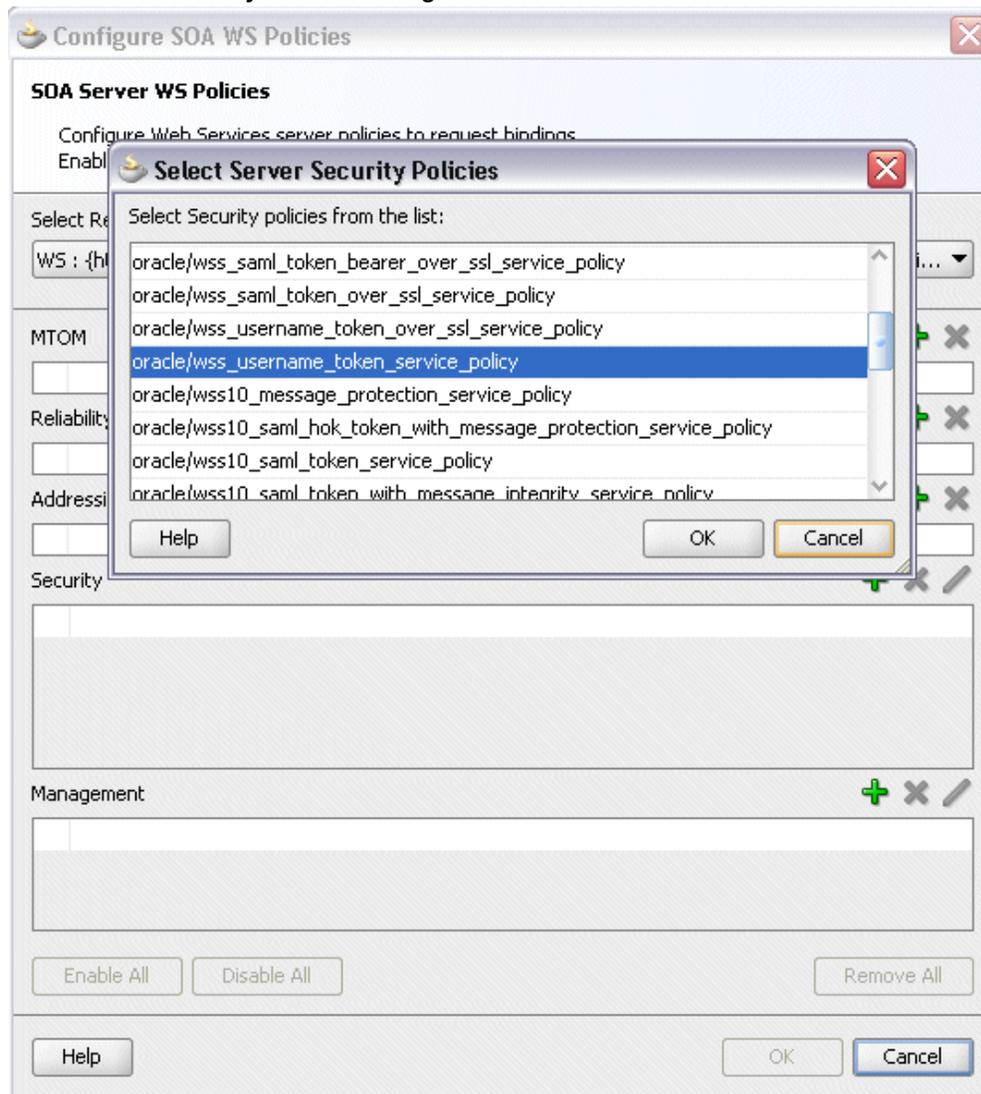
- TRANSACTION_SUBTYPE: 'PROCESS'
 - DOCUMENT_NUMBER: 'PO-4466-5'
 - PARTY_SIDE_ID: 'Example-01'
5. Click **Apply** and **OK** to complete the configuration of the Assign activity.

Configuring Web Service Policies

Use the following steps to add security policies at design time:

1. Navigate to SOA Content > Business Rules > composite.xml. Right click on the ONT__POI service and select "Configure WS Policies" from the drop-down list.
2. The Configure SOA WS Policies dialog appears.
In the Security section, click the **Add** icon (+). The Select Server Security Policies dialog appears.

Select Server Security Policies Dialog



Select 'oracle/wss_username_token_service_policy' and click **OK**.

The attached security policy is shown in the Security section.

A lock icon appears in the ONT__POI service of the composite.xml indicating that a security policy has been successfully attached.

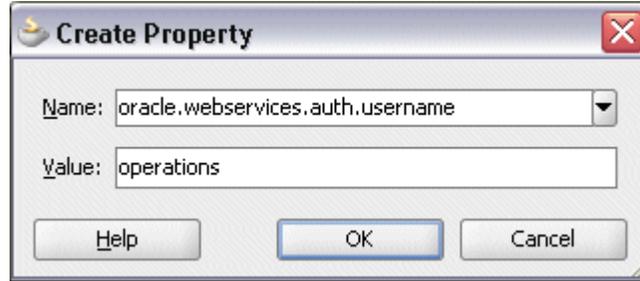
3. From the navigation menu, select **View > Property Inspector** to display the Property Inspector window for ONT__POI service component.

In the Properties section, click the **Add** icon (+) for binding properties. The Create Property dialog appears.

Enter 'oracle.webservices.auth.username' in the Name field and enter

'operations' as the value.

Create Property Dialog for Entering Username Property



Click **OK**.

4. Use the same approach by clicking the **Add** icon (+) again in the Properties section for binding properties. Enter 'oracle.webservices.auth.password' in the Name field. Enter the associated password for user 'operations' in the Value field.

Create Property Dialog for Entering Password Property



Click **OK**.

Both selected property names and values appear in the Properties section.

Click the Source tab of `composite.xml` and notice that the `oracle.webservices.auth.username` and `oracle.webservices.auth.password` property names and the associated values are added to the `ONT__POI` reference.

Verification of the Property Information in Composite.xml

```
30 <reference name="ONT_POI" ui:wSDLLocation="ONT_POI.wsdl">
31 <interface.wSDL interface="http://xmlns.oracle.com/apps/cln/soapprovider/xmlgateway/ont_poi/#wsdl.interface(ON
32 <binding.ws port="http://xmlns.oracle.com/apps/cln/soapprovider/xmlgateway/ont_poi/#wsdl.endpoint(ONT_POI_Ser
33 location="ONT_POI.wsdl" soapVersion="1.1">
34 <wsp:PolicyReference URI="oracle/wss_username_token_client_policy"
35 orawsp:category="security" orawsp:status="enabled"/>
36 <property name="oracle.webservices.auth.username" type="xs:string"
37 many="false" override="may">operations</property>
38 <property name="oracle.webservices.auth.password" type="xs:string"
39 many="false" override="may">password</property>
40 </binding.ws>
41 </reference>
```

Deploying and Testing the SOA Composite with BPEL Process at Runtime

To invoke the synchronous XML Gateway inbound map (PROCESS_PO_007) service from the BPEL client contained in the SOA composite, the SOA composite needs to be deployed to the Oracle WebLogic managed server. This can be achieved using Oracle JDeveloper. Once the composite is deployed, it can be tested from the Oracle Enterprise Manager Fusion Middleware Control Console.

Prerequisites

Before deploying the SOA composite with BPEL process using Oracle JDeveloper, you must have established the connectivity between the design-time environment and the runtime server. For information on how to configure the necessary server connection, see *Configuring Server Connection*, page B-1.

Note: If a local instance of the WebLogic Server is used, start the WebLogic Server by selecting Run > Start Server Instance from Oracle JDeveloper. Once the WebLogic Admin Server "DefaultServer" instance is successfully started, the <Server started in Running mode> and DefaultServer started message in the Running:DefaultServer and Messages logs should appear.

For the payload information, see *Sample Payload for Inbound Process Purchase Order XML Transaction*, page C-3.

Perform the following runtime tasks:

1. Deploy the SOA Composite with BPEL Process, page 5-22
2. Test the SOA Composite Application, page 5-25

Deploying the SOA Composite with BPEL Process

You must deploy the SOA composite with BPEL process (XMLGatewayInbound.bpel) that you created earlier before you can run it.

Note: Before deploying the with SOA composite 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.
- Edit the input file `order_data_xmlg.xml` by entering values for `<REFERENCEID>` and `<POID>` such as 'PO-4466-5'.

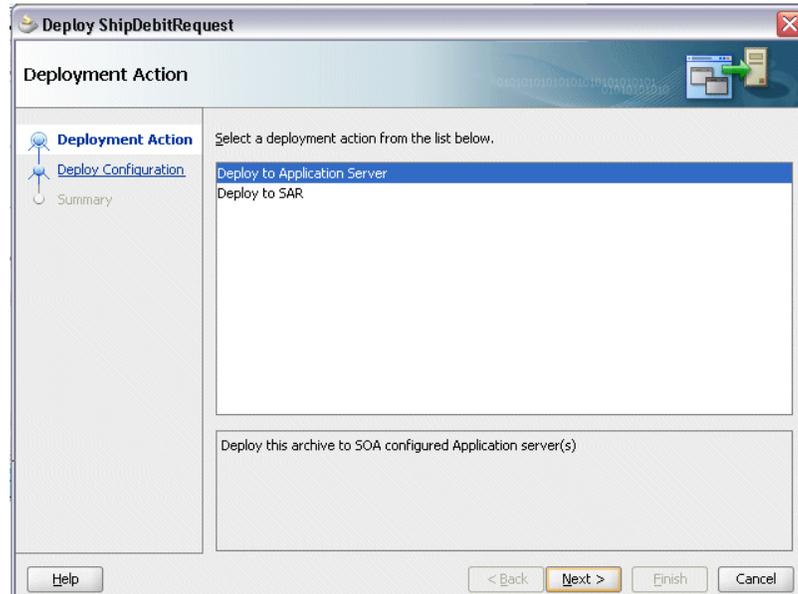
To deploy the SOA Composite with BPEL process:

1. In the Applications Navigator of JDeveloper, select the **XMLGInbound** project.
2. Right-click the project and select **Deploy > [project name] > [serverConnection]** from the menu.

For example, you can select **Deploy > XMLGInbound > SOAServer** to deploy the process if you have the connection set up appropriately.

Note: If this is the first time to set up the server connection, then the Deployment Action dialog appears. Select 'Deploy to Application Server' and click **Next**.

Deployment Action Dialog



In the Deploy Configuration dialog, ensure the following information is selected before clicking **Next** to add a new application server:

- New Revision ID: 1.0
- Mark composite revision as default: Select this checkbox.
- Overwrite any existing composites with the same revision ID: Select this checkbox.

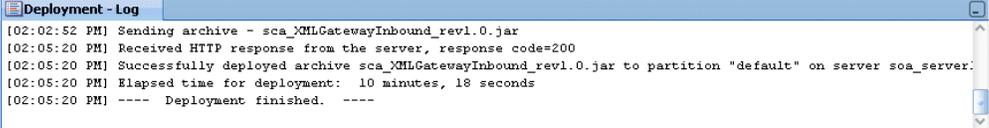
The steps to create a new Oracle WebLogic Server connection from JDeveloper are covered in Configuring Server Connection, page B-1.

3. In the Select Server dialog, select 'soa-server1' that you have established the server connection earlier. Click **Next**.
4. In the SOA Servers dialog, accept the default target SOA Server ('soa-server1') selection.
Click **Next** and **Finish**.
5. If you are deploying the composite for the first time from your Oracle JDeveloper session, the Authorization Request window appears. Enter username and password information specified during Oracle SOA Suite installation. Click **OK**.

6. Deployment processing starts. Monitor deployment process and check for successful compilation in the SOA - Log window.

Verify that the deployment is successful in the Deployment - Log window.

Deployment - Log Window



```
[02:02:52 PM] Sending archive - sca_XMLGatewayInbound_rev1.0.jar
[02:05:20 PM] Received HTTP response from the server, response code=200
[02:05:20 PM] Successfully deployed archive sca_XMLGatewayInbound_rev1.0.jar to partition "default" on server soa_server:
[02:05:20 PM] Elapsed time for deployment: 10 minutes, 18 seconds
[02:05:20 PM] ---- Deployment finished. ----
```

Testing the SOA Composite Application with BPEL Process

Once the BPEL process contained in the SOA composite application has been successfully deployed, you can manage and monitor the process from Oracle Enterprise Manager Fusion Middleware Control Console. You can also test the process and the integration interface by manually initiating the process.

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

To test the SOA composite application with BPEL process:

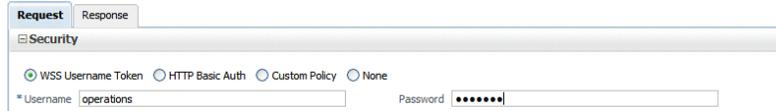
1. Navigate to Oracle Enterprise Manager Fusion Middleware Control Console (<http://<hostname>:<port>/em>). The login page appears.
2. Enter the user name and password information specified during the installation, and then click **Login** to log in to a farm. The composite (XMLGInbound) you deployed is displayed in the Applications Navigation tree.

You may need to select an appropriate target instance farm if there are multiple target Oracle Enterprise Manager Fusion Middleware Control Console farms.
3. From the Farm navigation pane, expand the SOA >soa-infra node in the tree to navigate through the SOA Infrastructure home page and menu to access your deployed SOA composite applications running on soa-infra managed server.

Click the XMLGInbound [1.0] link.
4. Click the Policies tab and notice that the 'oracle/wss_username_token_service_policy' policy you attached to the ONT__POI service binding earlier at the design time is now displayed here.
5. In the XMLGInbound [1.0] home page, click **Test**.
6. The Test Web Service page for initiating an instance appears.

Note: If the WS-Security credentials are not entered at design time, you can enter the credentials at runtime by selecting the WSS Username Token option in the Security section at the top of the Request tab. Enter 'operations' in the Username field and the associated password for user 'operations' in the Password field.

Test Web Service Page: Request Tab



7. Enter the input string (such as 'test') required by the process and click **Test Web Service** to initiate the process.

The test results appear in the Response tab upon completion.

8. Click the Instances tab. The SOA composite application instance ID, name, conversation ID, most recent known state of each instance since the last data refresh of the page are displayed.
9. Click your BPEL service component instance link (such as XMLGInbound) to display the Instances page where you can view the process details of the BPEL activities in the Audit Trail tab.

Click the Flow tab to check the BPEL process flow diagram. Click an activity of the process diagram to view the activity details and flow of the payload through the process.

Verifying Records in Oracle E-Business Suite

Once the BPEL process has been successfully initiated and completed, you can validate the transaction in Oracle E-Business Suite.

To Validate the Transaction 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 as a user who has 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

Transaction Monitor: Search

Select search criteria and press Go to view the report.

Inbound Messages
Processing Status

Outbound Messages
Generation Status
Delivery Status
Retry Status

Transaction Type Transaction Subtype

Source TP Location Code Trading Partner Name

Document ID Site Name

Party Type

From Date
To Date

Copyright (c) 1998, 2015, Oracle and/or its affiliates. All rights reserved. [Privacy Statement](#)

2. Clear From Date and To Date fields and enter 'PO-4466-5' in the Document ID field.
3. Select Customer as the Party Type. Click **Go**.
This retrieves XML inbound transaction 'PO-4466-5' in the Inbound Search Results region.
4. Confirm that the transaction 'PO-4466-5' has status 'SUCCESS'.

You can verify it by logging on to Oracle E-Business Suite with the Order Management Super User, Vision Operations (USA) responsibility. Select **Orders, Returns: Import Orders > Corrections**. The Find Orders window is displayed and select 'XML' from drop-down list in the Order Source field. The Corrections window is displayed with all the transactions with the XML order source.

Notice that 'PO-4466-5' is listed in the Orig_Sys_Document_Reference field.

Corrections Window to Verify the Transactions

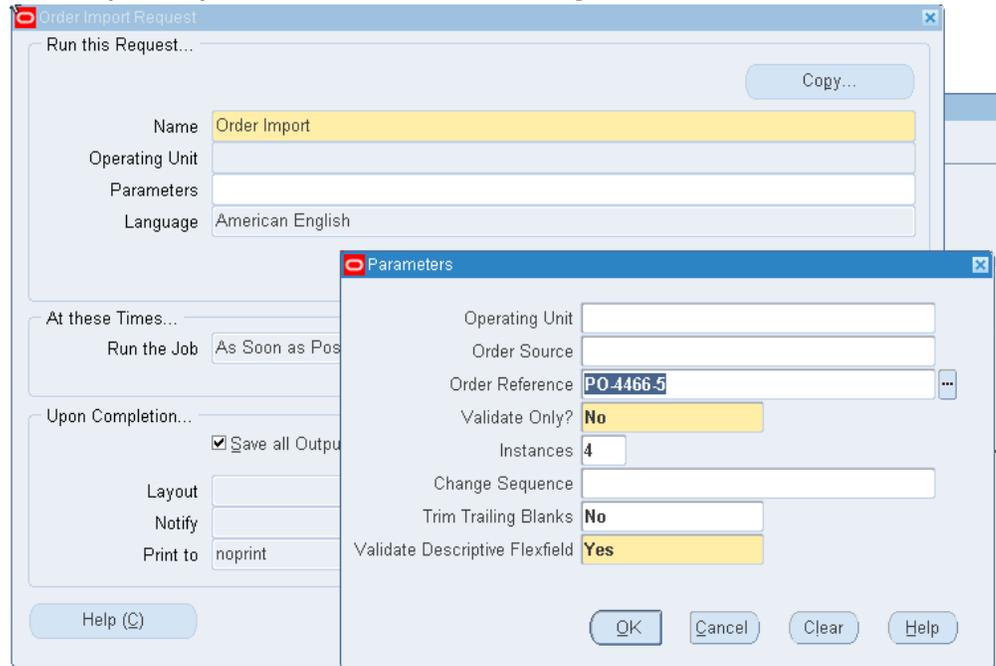
Order S	Orig Sys Docu	Change	Request ID	Operation Code	Order Number	Sold To Org ID
20	PO-4466-5	Change				
20	order_id_01	Change				
20	order_id_01	Change				

Payments Discounts Pricing Attributes Sales Credits
Actions Add Customers Errors Validate
Lines Import

To Import the Order to Oracle Order Management:

1. Log in to the Forms-based Oracle E-Business Suite as a user who has the Order Management Super User, Vision Operations (USA) responsibility.
2. Select **Orders, Returns : Import Orders > Order Import Request**. The Order Import Request form is displayed along with the Parameters dialog.
Select 'PO-4466-5' from the Order Reference drop-down list.

Order Import Request Form and Parameters Dialog



3. Click **OK** in the parameters dialog. The Order Import request name is populated automatically in the Import Request form.
4. Click **Submit** to submit the request. This displays a concurrent request number. Record the request number, but click **No** in the Decision dialog that you will not submit another request.
5. From the application menu, select **View > Requests** to open the Find Requests form.

6. Enter the request number you recorded earlier and click **Find**.

This would show the status of Order Import request. It should be success and the order should be created in Order Management application.

To Validate the Transaction in Oracle Order Management:

1. Log in to the Forms-based Oracle E-Business Suite as a user who has the Order Management, Super User responsibility.
2. Select **Order Returns > Sales Order**. The Sales Order form is displayed.
3. Search for an order by entering the order number in the Customer PO field (such as 'PO-4466-5'). The details of a newly created order appears.

Sales Orders Form

Sales Orders (Vision Operations) - 85182

Order Information Line Items

Default

Main Others

Customer		Order Number	
Customer Number		Order Type	Mixed
Customer PO	PO-4466-5	Date Ordered	04-SEP-2012 23:26:29
Customer Contact		Price List	Corporate
Ship To Location		Salesperson	
		Status	Entered
		Currency	USD
Bill To Location		Subtotal	26,156.40
		Tax	0.00
		Charges	150.00
		Total	26,306.40

[]

Actions Related Items Configurator Availability Book Order

Using XML Gateway Outbound Through Subscription Model

This section includes the following topics:

- Using XML Gateway Outbound Messages in Creating a New SOA Composite Application with BPEL Process at Design Time, page 5-30
- Deploying and Testing the SOA Composite Application with BPEL Process at Runtime, page 5-50

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 BPEL PM listens to the ECX_OUTBOUND queue for the message with the same correlation Id.

Oracle E-Business Suite Integrated SOA Gateway supports it through subscription model by dequeuing the message in the ECX_OUTBOUND queue to retrieve outbound data from Oracle E-Business Suite. The retrieved data can be passed to trading partners or consumers who subscribed to the message.

SOA Composite Application with 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_OAG72_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 the 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 the 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.

When the BPEL process has been successfully processed after deployment, the same order book reference ID (Customer PO) from the output XML file should be obtained 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 as a user who has 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.

XML Gateway Lookups Form

XML Gateway Lookups

Type:

Meaning:

Application:

Description:

Access Level:
 User
 Extensible
 System

Code	Meaning	Description	Tag	Effective Dates		Enabled
				From	To	
BPEL	BPEL	BPEL process		07-DEC-2006		<input checked="" type="checkbox"/>
HTTP	HTTP	HTTP Delivery		28-SEP-2000		<input checked="" type="checkbox"/>
HTTP-ATCH	Attachment Enabled H	Attachment Enabled HT		12-DEC-2002		<input checked="" type="checkbox"/>
HTTP-OXTA	Oracle Transport Agen	OXTA over HTTP		29-MAY-2000		<input checked="" type="checkbox"/>
HTTP-OXTAA1	Attachment Enabled C	Attachment Enabled OX		25-OCT-2002	12-DEC-2002	<input type="checkbox"/>
HTTP-WM	WebMethods using HI	HTTP with WebMethods		05-OCT-2000		<input checked="" type="checkbox"/>
HTTPS	HTTPS (Secure HTTP)	HTTPS Delivery		28-SEP-2000		<input checked="" type="checkbox"/>
HTTPS-ATCH	Attachment Enabled H	Attachment Enabled HT		12-DEC-2002		<input checked="" type="checkbox"/>
HTTPS-OMB	Oracle Message Broke	OMB using HTTPS		01-DEC-2000		<input type="checkbox"/>
HTTPS-OXTA	Oracle Transport Agen	OXTA over HTTPS		29-MAY-2000		<input checked="" type="checkbox"/>

5. Add a new record to the COMM_METHOD type by entering BPEL for the **Code** field and the **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.

In addition to having the correlation identifier set up correctly, you need to ensure the trading partner that you want to use has the Protocol Type field set to BPEL. For information on how to set up the trading partner with desired transactions, see *Manually test the SOA Composite Application*, page 5-53.

Trading Partner Setup Form

Transaction Type	Transaction SubType	Standard Code	External Transaction Type	External Transaction SubType	Direction	Map	Connection/Hub	Protocol Type
ONT	POA	OAG	PO	ACKNOWLEDG	OUT	ONT_3A4A_O	DIRECT	SMTP
ONT	CPO	OAG	PO	CANCEL	IN	ONT_3A9R_O		
ONT	POI	OAG	PO	PROCESS	IN	ONT_3A4R_O		
ONT	SSO	OAG	SALESORDE	SHOW	OUT	PROCESS_P	DIRECT	SMTP
ECX	CBODO	OAG	BOD	CONFIRM	OUT	ECX_CBODO	DIRECT	BPEL

SOA Composite Application with 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 SOA Composite Application with BPEL Process, page 5-34
Use this step to create a new SOA composite application with BPEL process called `XMLGOutbound.bpel`.
2. Create a Partner Link for AQ Adapter, page 5-35
Use this step to dequeue the event details from the `ECX_OUTBOUND` queue.
3. Add a Receive activity, page 5-44
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 5-46
This is to write PO acknowledgement details in an XML file as an output file.
5. Add an Invoke activity, page 5-48
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 5-49

Use the Assign activity to take the output from the Receive activity and to provide input to the Invoke activity.

For general information and how to create SOA composite applications using BPEL process service component, see the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite* for details.

Creating a New SOA Composite Application with BPEL Process

Use this step to create a new SOA composite application that will contain various BPEL process activities.

To create a new SOA composite application with BPEL process:

1. Open Oracle JDeveloper.
2. Click **New Application** in the Application Navigator. The "Create SOA Application - Name your application" page is displayed.
3. Enter an appropriate name for the application in the Application Name field and select **SOA Application** from the Application Template list.

Click **Next**. The "Create SOA Application - Name your project" page is displayed.

4. Enter an appropriate name for the project in the Project Name field. For example, XMLGatewayOutbound.
5. In the Project Technologies tab, select 'Web Services' and ensure that **SOA** is selected from the Available technology list to the Selected technology list.

Click **Next**. The "Create SOA Application - Configure SOA settings" page is displayed.

6. Select **Composite With BPEL Process** from the Composite Template list, and then click **Finish**. You have created a new application, and a SOA project. This automatically creates a SOA composite.

The Create BPEL Process page is displayed.

7. Leave the default **BPEL 1.1 Specification** selection unchanged. This creates a BPEL project that supports the BPEL 1.1 specification.

Enter an appropriate name for the BPEL process in the Name field. For example, XMLGOutbound.

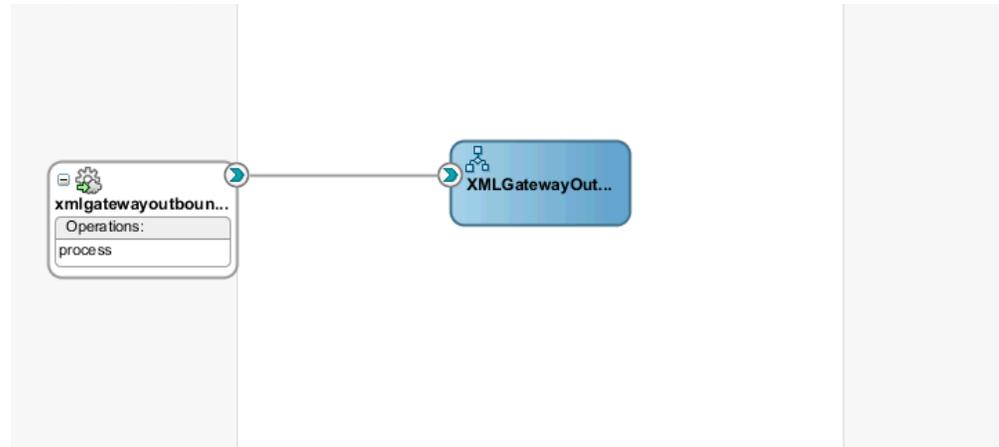
Select **Synchronous BPEL Process** in the Template field.

Select **required** from the Transaction drop-down list. Click **OK**.

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

8. Navigate to SOA Content > Business Rules and click `composite.xml` to view the composite diagram.

Composite Diagram



Double click on the XMLGOutbound component to open the BPEL process.

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_OAG72_OUT`) in the `ECX_OUTBOUND` queue.

To create a partner link for AQ Adapter:

1. In Oracle JDeveloper, drag and drop the **AQ Adapter** service from the **BPEL Services** list into the right Partner Link swim lane of the process diagram. The Adapter Configuration wizard welcome page appears.
2. Click **Next**. The Service Name dialog box appears.
3. Enter a service name in the Service Name dialog box, for example `GetAck`.

Service Name Dialog

The screenshot shows a dialog box titled "Adapter Configuration Wizard - Step 2 of 7". The main heading is "Service Name". Below the heading, it says "Enter a Service Name." and "Service Type: AQ Adapter". There is a text input field labeled "Service Name:" containing the text "GetAck". At the bottom of the dialog, there are four buttons: "Help", "< Back", "Next >", and "Finish". The "Next >" button is highlighted with a blue border.

4. Click **Next**. The Service Connection dialog box appears.
5. 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 appropriate username and password information 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.
 6. 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.
 7. Click **Next** to open Adapter Interface dialog box.
Select the **Define from operation and schema (specified later)** radio button and click **Next**. The Operation dialog box appears.
 8. Select the **Dequeue** radio button in the Operation Type field. 'Dequeue' is also populated in the Operation Name field.

Operation Dialog

Operation

The AQ Adapter supports three operations. There is a Dequeue operation that polls for incoming messages from a queue, an Enqueue operation that puts outgoing messages on a queue, and a Enqueue/Dequeue operation that puts outgoing messages on a queue and expects response messages on a queue. Specify the Operation type and Operation Name. Only one operation per Adapter Service may be defined using this wizard.

Operation Type: Dequeue
 Enqueue
 Enqueue/Dequeue

Operation Name: Dequeue

Help < Back Next > Finish Cancel

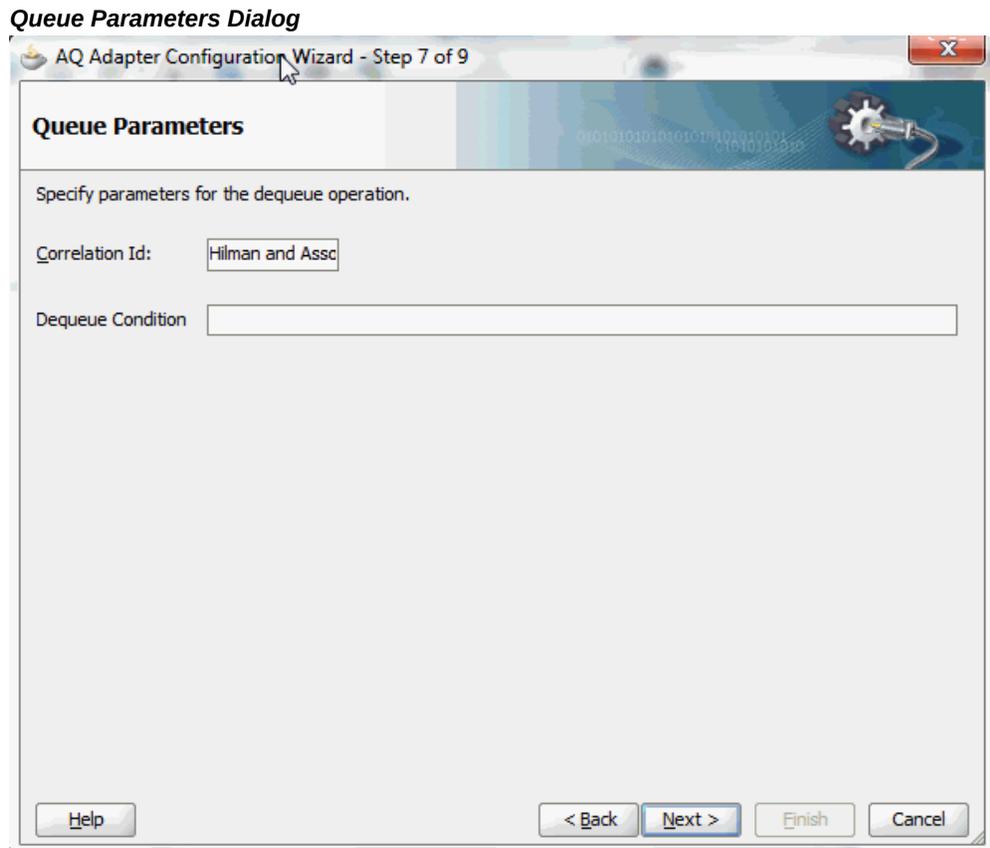
9. Click **Next** to open the Queue Name dialog box.

Select 'APPLSYS' as the Database Schema field. Enter 'ECX_OUTBOUND' as the Queue Name field.

Queue Name Dialog

The screenshot shows a dialog box titled "Queue Name" within the "AQ Adapter Configuration Wizard - Step 6 of 7". The dialog has a blue header bar with a gear icon and binary code. Below the header, there is a text instruction: "Specify the database schema and the queue to be used for the service. Use the Browse button to find the queue in the specified schema." To the right of this text is a "Browse" button. Below the instruction is a section titled "Queue Information" containing two input fields: "Database Schema:" with a dropdown menu showing "APPLSYS" and "Queue Name:" with a text box containing "ECX_OUTBOUND". At the bottom of the dialog, there are four buttons: "Help", "< Back", "Next >", and "Cancel".

10. Click **Next** to open the Queue Parameters dialog box.



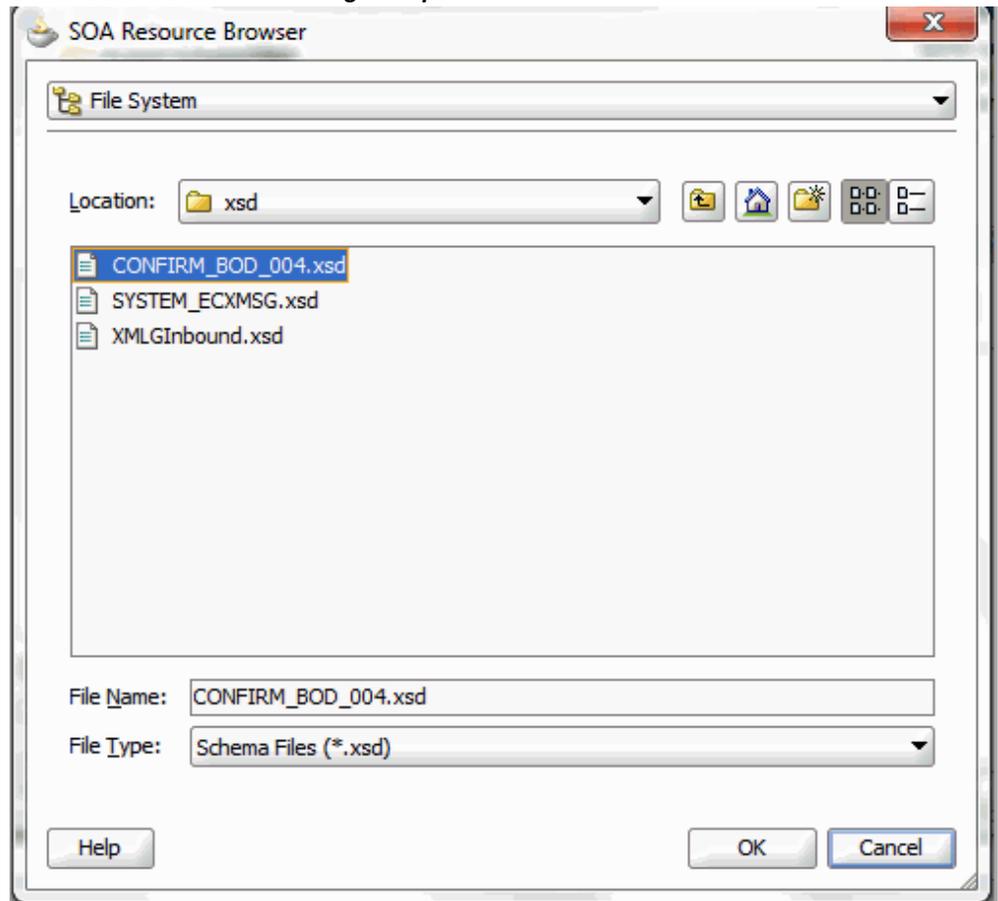
Enter a unique customer name, such as 'Hilman and Associates', in the Correction Id field. Click **Next**. The Messages dialog box opens.

11. Click **Browse for schema file** to open the Type Chooser window.

Click **Import Schema Files** on the top right corner of the Type Chooser window. This opens the Import Schema File pop-up window.

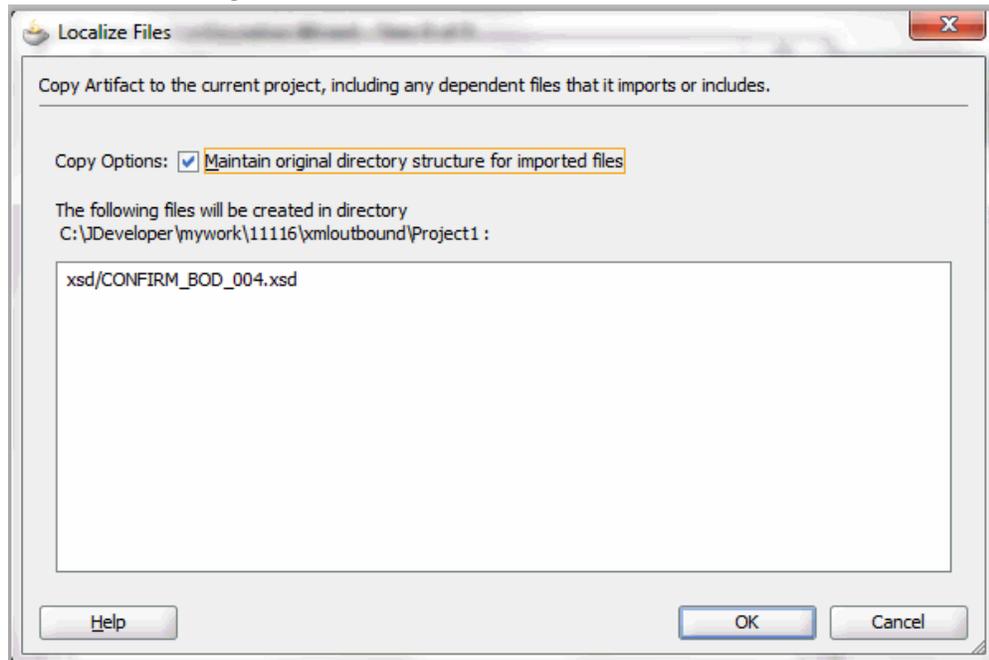
Click the **Browse Resources...** icon to display the SOA Resource Browser window. Select the xsd folder as the location and CONFIRM_BOD_004 .xsd file from the folder.

SOA Resource Browser Dialog to Import a Desired Schema



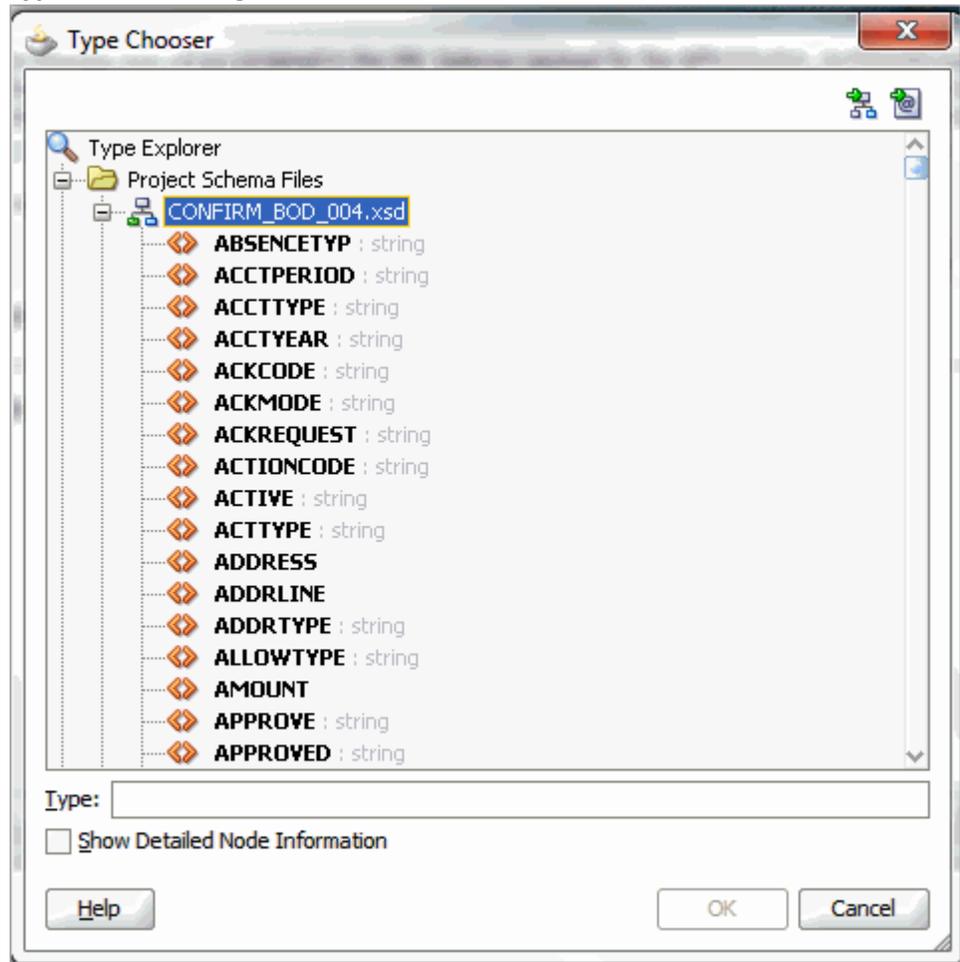
Click **OK**. The Localize Files dialog box opens. Click **OK**.

Localize Files Dialog



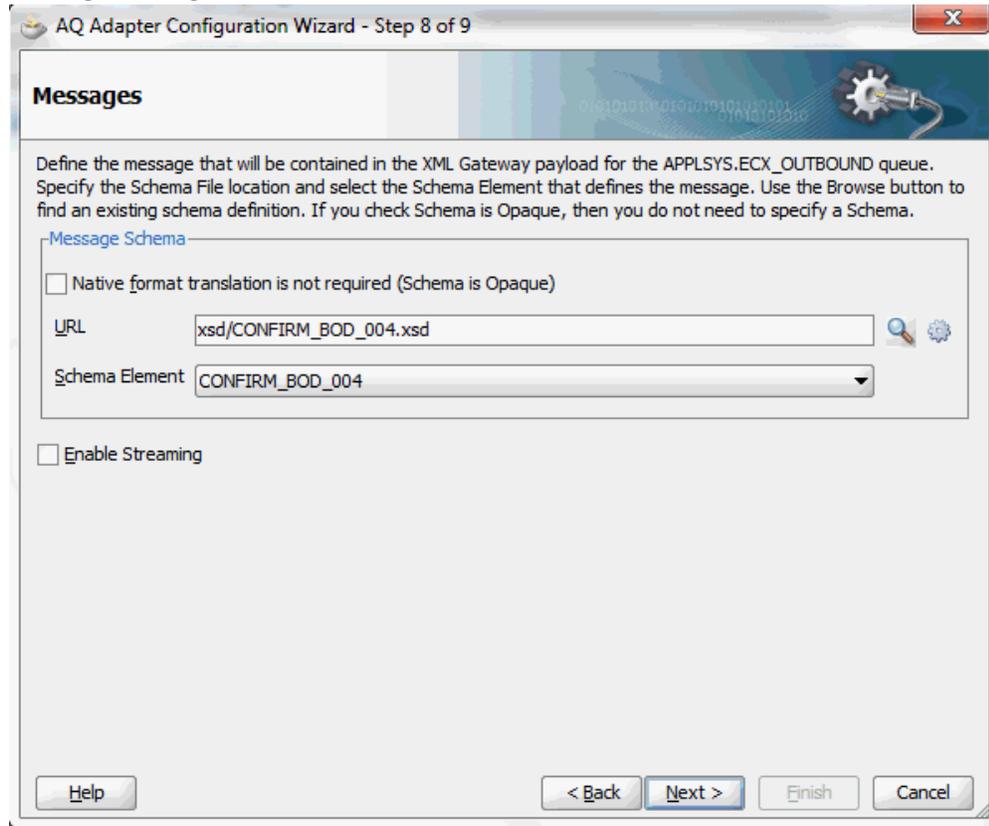
In the Type Chooser window, the selected CONFIRM_BOD_004.xsd file is displayed. Scroll down to select the CONFIRM_BOD_004 schema from CONFIRM_BOD_004.xsd. Click **OK**.

Type Chooser Dialog to Select a Desired Schema



The Messages dialog box is automatically displayed with the selected schema location and schema element.

Messages Dialog with Selected Schema



12. Click **Next** to proceed to the Finish dialog box to confirm that you have finished defining the AQ Adapter for the GetAck service.
13. 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.

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 Oracle JDeveloper, expand the **BPEL Constructs** from the Component Palette.

Drag and drop the **Receive** activity into the center swim lane of the process diagram.

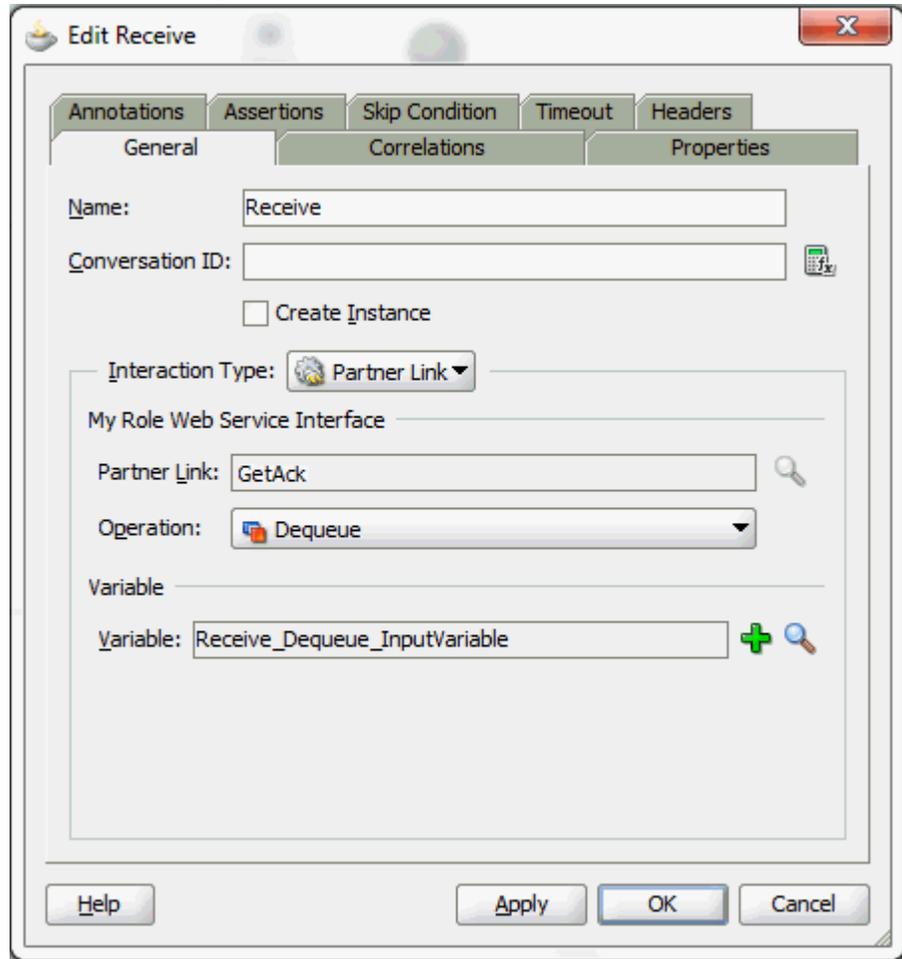
2. Link the **Receive** activity to the GetAck partner link. The Receive activity will take event data from the partner link. The Edit Receive dialog box appears.

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.

Edit Receive Dialog



5. Click **Apply** and **OK** to finish configuring the Receive activity.

The Receive activity appears in the BPEL process diagram.

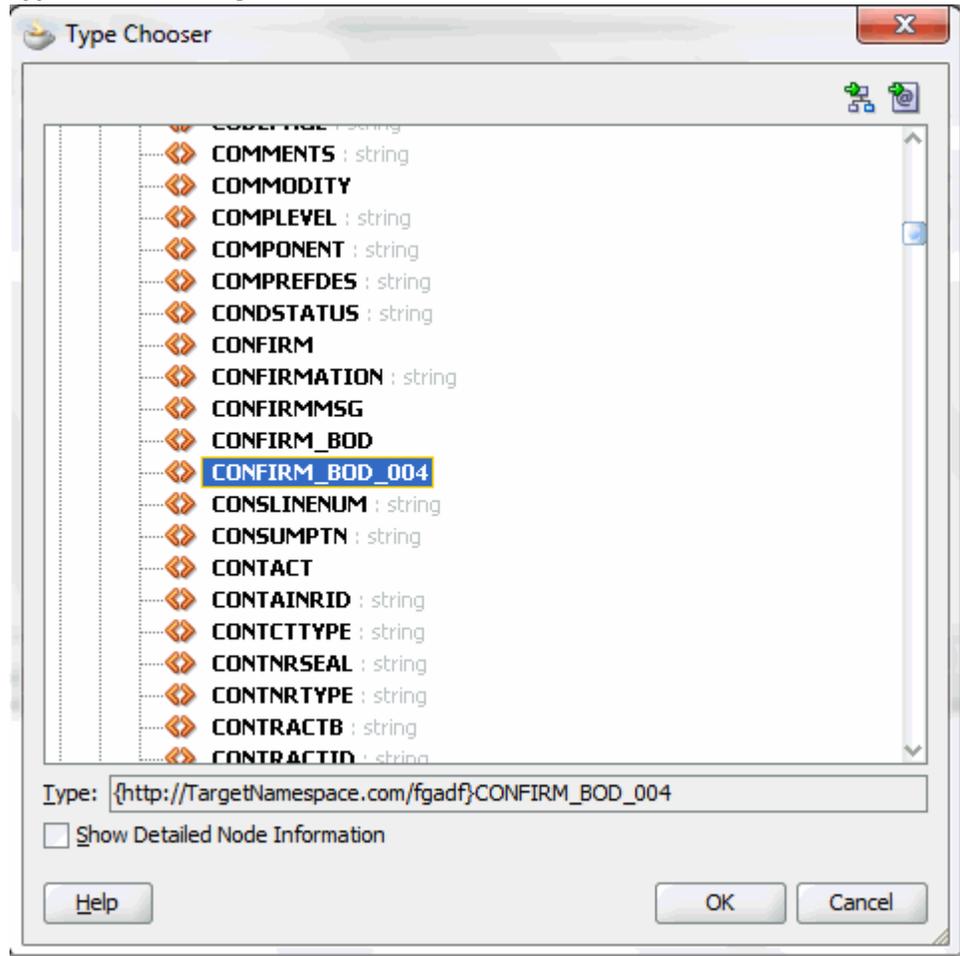
Adding a Partner Link for File Adapter

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:

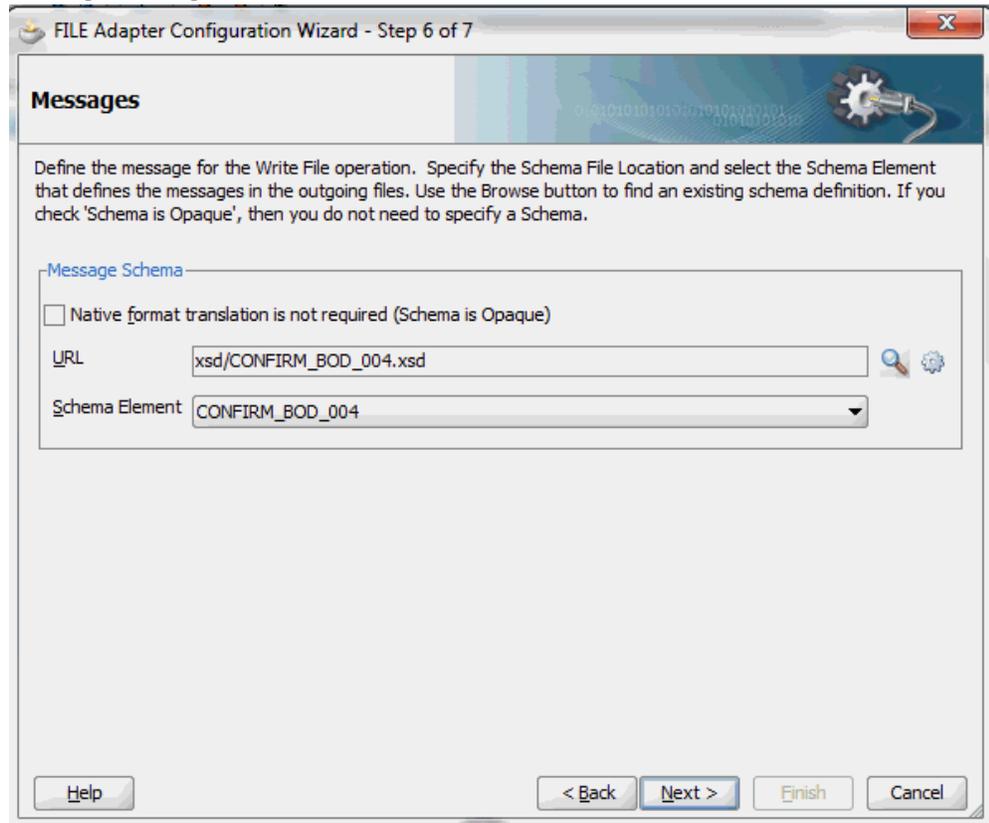
1. In Oracle JDeveloper, drag and drop the **File Adapter** service from the **BPEL Services** list into the right Partner Link swim lane of the process diagram. The Adapter Configuration wizard appears.
2. Click **Next**. The Service Name dialog box appears.
3. Enter a name for the File Adapter service, such as `WriteAck`.
4. Click **Next** and the Adapter Interface dialog box appears.
Select the **Define from operation and schema (specified later)** radio button and click **Next**. The Operation dialog box appears.
5. Specify the operation type, for example **Write File**. This automatically populates the **Operation Name** field.
6. Click **Next** to access the File Configuration dialog box.
7. 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 `PO_%yMMddHHmmss%.xml`.
8. Confirm the default write condition: Number of Messages Equals 1. Click **Next**. The Messages dialog box appears.
9. Select the **Browse** checkbox to locate the schema location and schema element.
The Type Chooser dialog box appears. Expand the **Project Schema Files > CONFIRM_BOD_004.xsd** and select `CONFIRM_BOD_004`.

Type Chooser Dialog



Click **OK** in the Type Chooser window to populate the selected schema location and element in the Messages dialog box.

Messages Dialog with Selected Schema



10. 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 Oracle JDeveloper, expand the **BPEL Constructs** from the Component Palette. Drag and drop the **Invoke** activity into the center swim lane 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.
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.

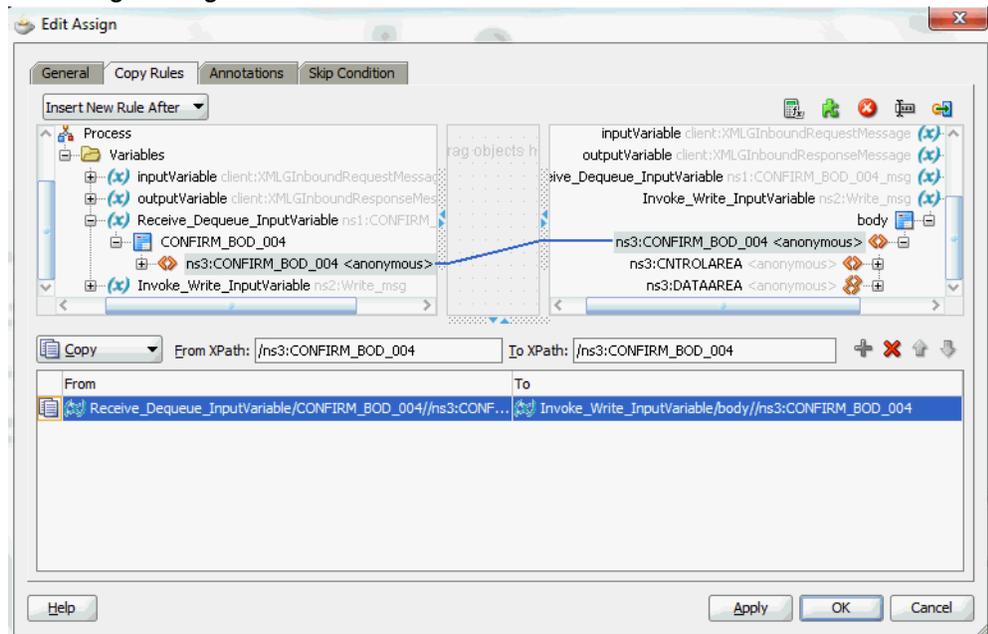
Adding an Assign Activity

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 Oracle JDeveloper, expand the **BPEL Constructs** from the Component Palette. Drag and drop the **Assign** activity from the Component Palette into the center swim lane 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. Select the Copy Rules tab and expand the source and target trees:
 - In the From navigation tree, navigate to **Variables > Process > Variables > Receive_DEQUEUE_InputVariable** and select **CONFIRM_BOD_004**. The XPath field contains your selected entry.
 - In the To navigation tree, navigate to **Variables > Process > Variables > Invoke_Write_InputVariable > body** and select **CONFIRM_BOD_004**. The XPath field contains your selected entry.

Edit Assign Dialog



Drag the source node (CONFIRM_BOD_004) to connect to the target node (CONFIRM_BOD_004) that you just identified. This creates a line that connects the source and target nodes. The copy rule is displayed in the From and To sections at the bottom of the Edit Assign dialog box.

4. Click **Apply** and then **OK** in the Edit Assign dialog box to complete the configuration of the **Assign** activity.

Deploying and Testing the SOA Composite Application with BPEL Process at Runtime

After creating a SOA composite application with BPEL process for XML Gateway outbound message map, you need to deploy it to the Oracle WebLogic managed server. This can be achieved using Oracle JDeveloper. Once the composite is deployed, it can be tested from the Oracle Enterprise Manager Fusion Middleware Control Console.

Prerequisites

Before deploying the SOA composite with BPEL process using Oracle JDeveloper, you must have established the connectivity between the design-time environment and the runtime server. For information on how to configure the necessary server connection, see *Configuring Server Connection*, page B-1.

Note: If a local instance of the WebLogic Server is used, start the WebLogic Server by selecting **Run > Start Server Instance** from Oracle JDeveloper. Once the WebLogic Admin Server "DefaultServer" instance

is successfully started, the <Server started in Running mode> and DefaultServer started message in the Running:DefaultServer and Messages logs should appear.

Perform the following runtime tasks:

1. Deploy the SOA Composite with BPEL Process, page 5-51
2. Manually Test the SOA Composite Application, page 5-53

Deploying the SOA Composite Application with BPEL Process

You must deploy the SOA composite application with BPEL process (XMLGOutbound.bpel) that you created earlier before you can run it.

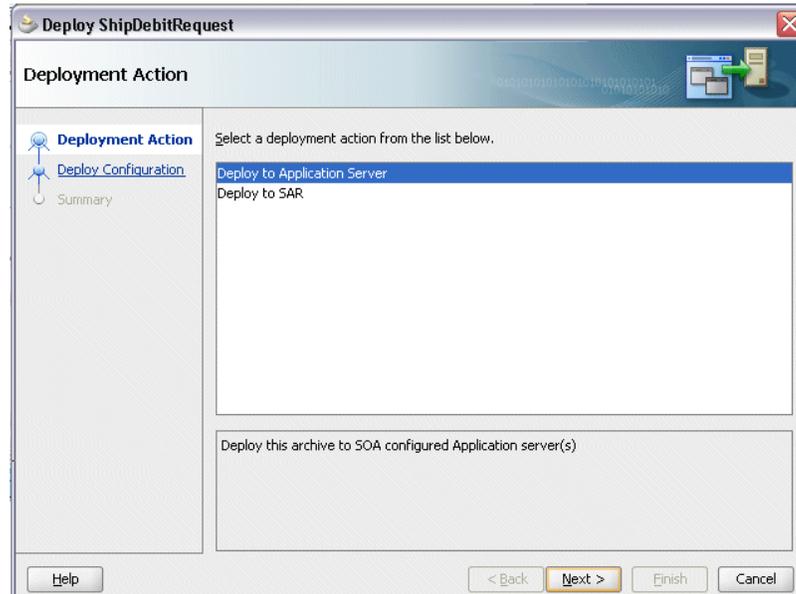
To deploy the SOA composite application BPEL process:

1. In the Applications Navigator of Oracle JDeveloper, select the **XMLGOutbound** project.
2. Right-click the project and select **Deploy > [project name] > [serverConnection]** from the menu.

For example, you can select **Deploy > XMLGInbound > SOAServer** to deploy the process if you have the connection appropriately.

Note: If this is the first time to set up the server connection, then the Deployment Action dialog appears. Select 'Deploy to Application Server' and click **Next**.

Deployment Action Dialog



In the Deploy Configuration dialog, ensure the following information is selected before clicking **Next** to add a new application server:

- New Revision ID: 1.0
- Mark composite revision as default: Select this checkbox.
- Overwrite any existing composites with the same revision ID: Select this checkbox.

The steps to create a new Oracle WebLogic Server connection from Oracle JDeveloper are covered in Configuring Server Connection, page B-1.

3. In the Select Server dialog, select 'soa-server1' that you have established the server connection earlier. Click **Next**.
4. In the SOA Servers dialog, accept the default target SOA Server ('soa-server1') selection.
Click **Next** and **Finish**.
5. If you are deploying the composite for the first time from your Oracle JDeveloper session, the Authorization Request window appears. Enter username and password information specified during Oracle SOA Suite installation. Click **OK**.

6. Deployment processing starts. Monitor deployment process and check for successful compilation in the SOA - Log window.

Verify that the deployment is successful in the Deployment - Log window.

Testing the SOA Composite Application with BPEL Process

Once the BPEL process contained in the SOA composite application has been successfully deployed, you can manage and monitor the process from Oracle Enterprise Manager Fusion Middleware Control Console. You can log in 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 Oracle Enterprise Manager Fusion Middleware Control 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 SOA composite application with BPEL process:

1. Log in to Oracle E-Business Suite as a user who has 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 window.

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_OAG72_OUT
- Connection / Hub: DIRECT
- Protocol Type: BPEL
- Username: 'operations'
- Password: Enter the associated password for the user 'operations' twice.
- Protocol Address: 'http://us.example.com'
Protocol Address is the complete URL where the XML document can be posted.
- Source Trading Partner Location Code: Example-01

Trading Partner Setup Form

Transaction Type	Transaction SubType	Standard Code	External Transaction Type	External Transaction SubType	Direction	Map	Connection/Hub	Protocol Type
ONT	POA	OAG	PO	ACKNOWLEDG	OUT	ONT_3A4A_O	DIRECT	SMTP
ONT	CPO	OAG	PO	CANCEL	IN	ONT_3A9R_O		
ONT	POI	OAG	PO	PROCESS	IN	ONT_3A4R_O		
ONT	SSO	OAG	SALESORDE	SHOW	OUT	PROCESS_P	DIRECT	SMTP
ECX	CBODO	OAG	BOD	CONFIRM	OUT	ECX_CBODO	DIRECT	BPEL

5. Save the trading partner details.

To successfully generated PO Acknowledgement, perform the following setup tasks in Order Management and then manually book the order:

1. 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.

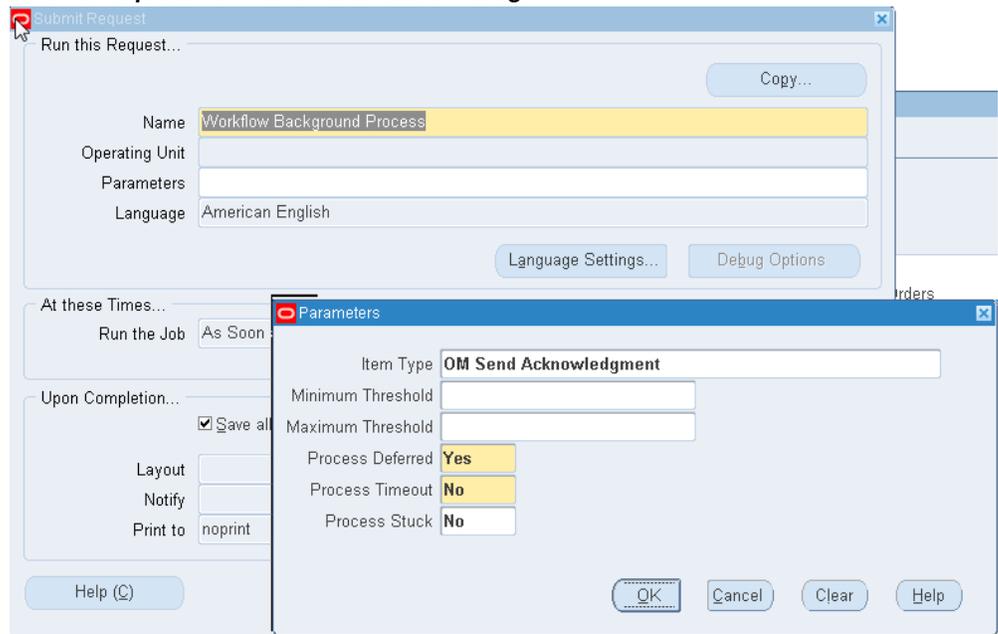
2. Search on the 'Example Inc' in the Name field and click **Go**.
3. Select the Business World with the following information from the search results:
 - Name: Example Inc.
 - Registry ID: 1004
 - Address: 401 Island Parkway Redwood Shores, CA 94065
4. Select the account with the following information:
 - Account Number: 1004
 - Account Description: Example Inc.
 - Status: Active
5. Click the **Details** icon to open the Update Account: 1608 page.
6. Locate the address with 401 Island Parkway Redwood City, CA 94065 and click the **Details** icon to open the Site page of your selected account.
7. In the Site Details tab, the Account Site Details region, enter 'Example-01' in the EDI Location field.
8. In the Business Purposes tab, create a new row with the following values:
 - Purpose: Sold To
 - Check on the 'Primary' checkbox

Click **Apply**.

Use the following steps to generate acknowledgement for the order that you have created for the XML Gateway Inbound service.

1. Log in to Oracle E-Business Suite as a user who has the same Order Management Super User, Vision Operations (USA) responsibility. Select **Order Returns > Sales Order** to open the Sales Orders form.
2. Retrieve the order that you have created earlier in the XML Gateway inbound service by entering the order ID 'PO-4466-5' in the Customer PO field.
3. Click **Book Order** to book the order.

Submit Request Form and Parameters Dialog



- Name: Workflow Background Process
 - Enter the following parameters:
 - Item Type: OM Send Acknowledgement
 - Process Deferred: Yes
 - Process Timeout: No
 - Process Stuck: No
 - Click **OK**.
7. Click **Submit** to submit the 'send acknowledgement' request.
 8. View your request by entering the request ID to ensure its status is 'Success'.

Validation Using Oracle Enterprise Manager Fusion Middleware Control Console

Log in to Oracle Enterprise Manager Fusion Middleware Control Console (<http://<servername>:<portname>/em>) to confirm that the XMLGOutbound process has been deployed. This process is continuously polling the ECX_OUTBOUND queue for purchase order acknowledgement.

From the Farm navigation pane, expand the SOA > soa-infra node in the tree to navigate through the SOA Infrastructure home page and menu to access your deployed

SOA composite applications running on `soa-infra` managed server. Click the XMLGOutbound [1.0] link.

Click **Test**. The Test Web Service page for initiating an instance appears. Enter the input string required by the process and click **Test Web Service** to initiate the process. The test results appear in the Response tab upon completion. Click your BPEL service component instance link (such as XMLGOutbound) to display the Instances page where you can view the process details of the BPEL activities in the Audit Trail tab.

Click the Flow tab to check the BPEL process flow diagram. Click an activity of the process diagram to view the activity details and flow of the payload through the process. For example, click **Receive** activity. Click the **view xml document** link to open the received XML file. Note the Reference ID such as Customer PO.

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 'PO_060719175318.xml').

Open the 'PO_060719175318.xml' file and search for ORIGREF element. Its value should be 'PO-4466-5' (the order booked) for which the acknowledgement is generated.

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 processed 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 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 SOA composite application with BPEL process in Oracle JDeveloper to include the subscribed event at design time and update application data if needed at runtime.

To better understand how the subscription model works for business events, detailed tasks at design time and runtime are included in this chapter. For the example described in the following sections, Oracle JDeveloper 11g (11.1.1.6.0) is used as a design-time tool to create a SOA composite application with BPEL process, and Oracle SOA Suite 11g (11.1.1.6.0) is used for the process deployment.

Note: While using Oracle JDeveloper with other Oracle Fusion Middleware components (such as Oracle SOA Suite), to enable SOA technologies, you need to manually download Oracle SOA Suite

Composite Editor, an Oracle JDeveloper extension for SOA technologies. For more information on installing additional Oracle Fusion Middleware design-time components, see the *Oracle Fusion Middleware Installation Guide for Oracle JDeveloper*.

Using a Business Event in Creating a SOA Composite Application with BPEL Process at Design Time

SOA Composite Application with BPEL Process Scenario

Take a PO XML Raise business event as an example to explain the SOA composite application with BPEL process creation.

When a purchase order is created and approved, a Purchase Order Approved business event `oracle.apps.po.event.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 written in XML file as an output file.

When the BPEL process has been successfully processed after deployment, the same purchase order information should be obtained from the output file once a purchase order is approved.

Prerequisites to Create a SOA Composite Application with BPEL Process Using a Business Event

An integration administrator 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 has been successfully created.

To subscribe to a business event, the administrator 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.

SOA Composite Application with 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 SOA Composite Application with BPEL Process, page 6-3
Use this step to create a new SOA composite application with BPEL process called `GetPurchaseOrder.bpel`.
2. Create a Partner Link for AQ Adapter, page 6-5
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 6-19
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 6-20
Use the Assign activity to take the output from the Receive activity and to provide input to the Invoke activity.

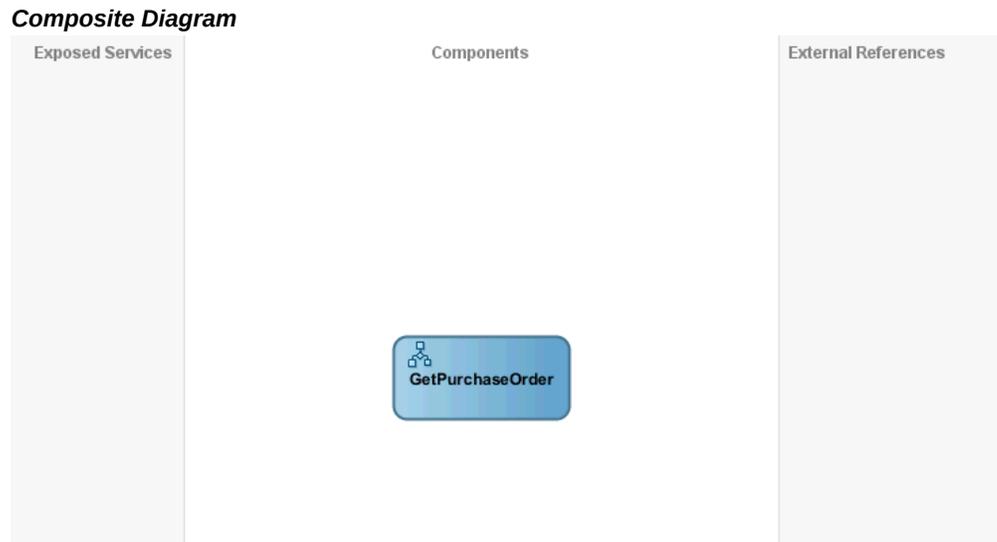
For general information and how to create SOA composite applications using BPEL process service component, see the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite* for details.

Creating a New SOA Composite Application with BPEL Process

Use this step to create a new SOA composite application with BPEL process that will contain various BPEL process activities.

To create a new SOA composite application with BPEL process:

1. Open Oracle JDeveloper.
2. Click **New Application** in the Application Navigator.
The "Create SOA Application - Name your application" page is displayed.
3. Enter an appropriate name for the application in the Application Name field and select **SOA Application** from the Application Template list.
Click **Next**. The "Create SOA Application - Name your project" page is displayed.
4. Enter an appropriate name for the project in the Project Name field, for example, `GetPOProject`.
5. In the Project Technologies tab, select 'Web Services' and ensure that **SOA** is selected from the Available technology list to the Selected technology list.
Click **Next**. The "Create SOA Application - Configure SOA settings" page is displayed.
6. Select **Composite With BPEL Process** from the Composite Template list, and then click **Finish**. You have created a new application, and a SOA project. This automatically creates a SOA composite.
The Create BPEL Process page is displayed.
7. Leave the default **BPEL 1.1 Specification** selection unchanged. This creates a BPEL project that supports the BPEL 1.1 specification.
Enter an appropriate name, for example `GetPurchaseOrder`, for the BPEL process in the Name field.
Select **Define Service Later** in the **Template** field. Click **OK**.
An empty BPEL process is created. The required source files including `bpel` and `wsdl`, using the name you specified (for example, `GetPurchaseOrder.bpel` and `GetPurchaseOrder.wsdl`) and `composite.xml` are also generated.
8. Click **Finish**.
9. Navigate to SOA Content > Business Rules and double click `composite.xml` to view the composite diagram.



Double click on the `GetPurchaseOrder` component to open the BPEL process.

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.event.xmlpo` event.

To create a partner link for AQ Adapter to dequeue the event subscription:

1. In Oracle JDeveloper, drag and drop the **AQ Adapter** service from the **BPEL Services** list into the right Partner Link swim lane of the process diagram. The Adapter Configuration wizard welcome page appears.
2. Click **Next**. The Service Name dialog box appears.
3. Enter a service name in the Service Name dialog box, for example `GetPO`.
4. Click **Next**. The Service Connection dialog box appears.
5. 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 appropriate user name and password information 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.

6. 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.

7. Click **Next** to open Adapter Interface dialog box.

Select the **Define from operation and schema (specified later)** radio button and click **Next**. The Operation dialog box appears.

8. Select the **Dequeue** radio button in the Operation Type field. 'Dequeue' is also populated in the Operation Name field.

Operation Dialog

The screenshot shows a dialog box titled "Adapter Configuration Wizard - Step 5 of 7". The main heading is "Operation". Below the heading, there is a paragraph of text: "The AQ Adapter supports three operations. There is a Dequeue operation that polls for incoming messages from a queue, an Enqueue operation that puts outgoing messages on a queue, and a Enqueue/Dequeue operation that puts outgoing messages on a queue and expects response messages on a queue. Specify the Operation type and Operation Name. Only one operation per Adapter Service may be defined using this wizard." Below this text, there are three radio button options for "Operation Type": "Dequeue" (which is selected), "Enqueue", and "Enqueue/Dequeue". Below the radio buttons, there is a text input field for "Operation Name" containing the text "Dequeue". At the bottom of the dialog, there are four buttons: "Help", "< Back", "Next >", and "Cancel".

9. Click **Next** to open the Queue Name dialog box.

Select 'APPS' in the Database Schema field. Enter 'WF_BPEL_Q' in the Queue Name field.

Queue Name Dialog

The screenshot shows a dialog box titled "Queue Name" from the "AQ Adapter Configuration Wizard - Step 6 of 9". The dialog has a header bar with a gear icon and a close button. Below the header, there is a decorative banner with binary code and a gear icon. The main content area contains the following text: "Specify the database schema and the queue to be used for the service. Use the Browse button to find the queue in the specified schema." Below this text is a "Browse" button. Underneath is a section titled "Queue Information" which contains a "Database Schema:" dropdown menu with "APPS" selected, and a "Queue Name:" text input field with "WF_BPEL_Q" entered. At the bottom of the dialog, there are four buttons: "Help", "< Back", "Next >", "Finish", and "Cancel".

10. Click **Next** to open the Queue Parameters dialog box.

Queue Parameters Dialog

The screenshot shows a dialog box titled "Queue Parameters" within the "Adapter Configuration Wizard - Step 7 of 9". The dialog has a light blue header with a gear icon and binary code. The main area is white with a light blue background. It contains the following fields:

- Consumer:** A text box containing "ISGDemo".
- Correlation Id:** An empty text box.
- Message Selector Rule:** A text box containing "tab.user_data.geteventname()='oracle.apps.po.event.xmlpo'".
- Dequeue Condition:** An empty text box.

At the bottom, there are four buttons: "Help", "< Back", "Next >", "Finish", and "Cancel".

Enter the following information:

- Enter a 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.event.xmlpo'`).

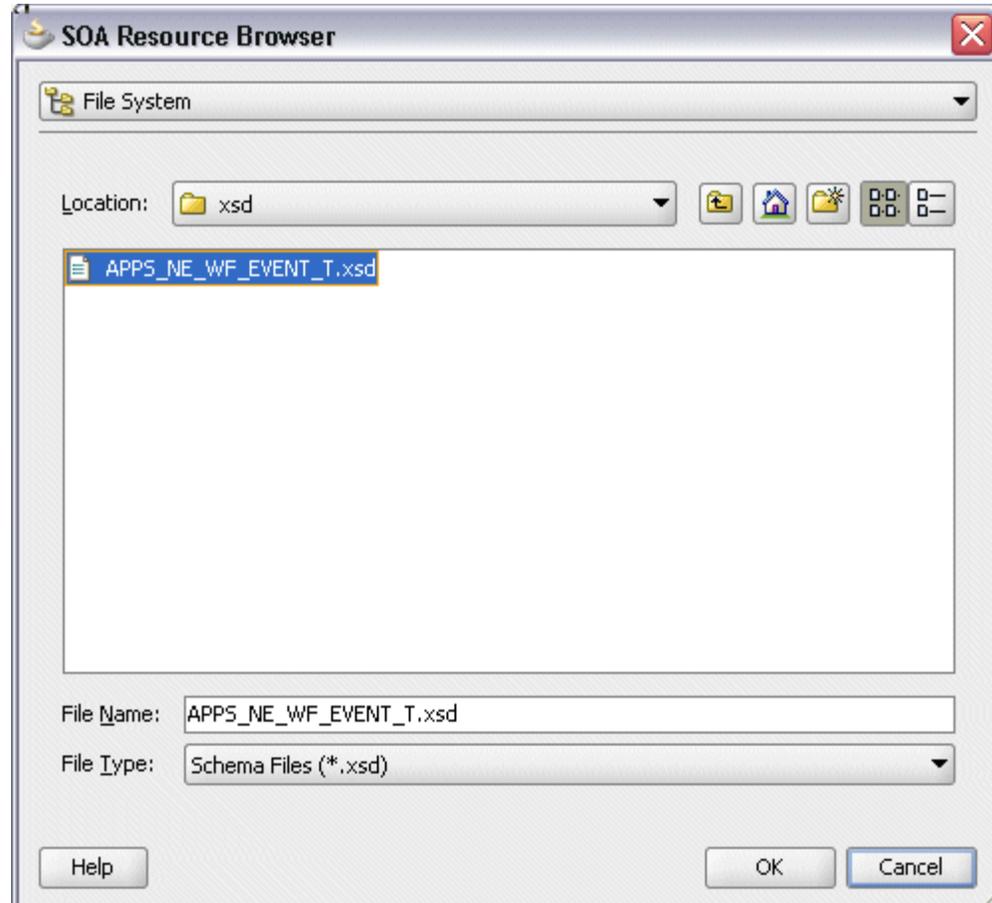
11. Click **Next**. The Messages dialog box opens where you can define the message that will be contained in the Business Event System payload for the APPS.WF_BPEL_Q queue.

Click **Browse for schema file** to open the Type Chooser window.

Click **Import Schema Files** on the top right corner of the Type Chooser window. This opens the Import Schema File pop-up window.

Click the **Browse Resources..** icon to display the SOA Resource Browser window. Select the `xsd` folder as the location and `APPS_NE_WF_EVENT_T` file from the folder.

SOA Resource Browser Window

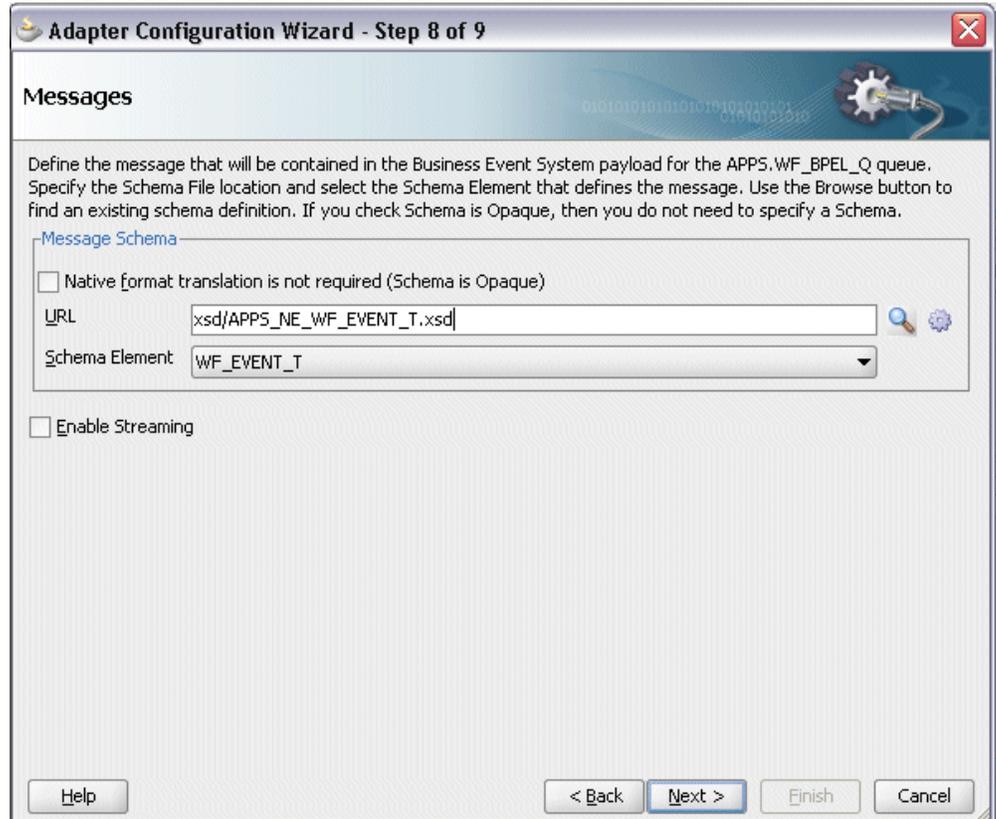


Click **OK** twice.

In the Type Chooser window, expand the Project Schema Files and select `WF_EVENT_T` schema from the `APPS_NE_WF_EVENT_T.xsd`. Click **OK**.

The selected `APPS_NE_WF_EVENT_T.xsd` is displayed as URL and the `WF_EVENT_T` element is selected as Schema Element.

Messages Dialog with Selected Schema



12. Click **Next** to proceed to the Finish dialog box to confirm that you have finished defining the AQ Adapter for the GetPO service.
13. Click **Finish**. The wizard generates the WSDL file corresponding to the GetPO 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 GetPO 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 Oracle JDeveloper, expand the **BPEL Constructs** from the Component Palette.

Drag and drop the **Receive** activity into the center swim lane of the process diagram.

2. Link the Receive activity to the GetPO 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.
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. Click **Apply** and then **OK** to finish configuring the Receive activity.

Edit Receive Dialog

The screenshot shows the 'Edit Receive' dialog box with the following configuration:

- Name:** Receive_PO
- Conversation ID:** (empty)
- Create Instance
- Interaction Type:** Partner Link
- My Role Web Service Interface:**
 - Partner Link:** GetPO
 - Operation:** DEQUEUE
- Variable:** Receive_PO_DEQUEUE_InputVariable

Buttons at the bottom: Help, Apply, OK, Cancel.

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 Oracle JDeveloper, drag and drop the **File Adapter** service from the **BPEL Services** list into the right Partner Link swim lane of the process diagram. The Adapter Configuration wizard appears.
2. Click **Next**. The Service Name dialog box appears.
3. Enter a name for the file adapter service, such as `WritePurchaseOrder`.
4. Click **Next**. The Adapter Interface dialog box appears.
5. Select the **Define from operation and schema (specified later)** radio button and click **Next**. The Operation dialog box appears.

Operation Dialog

The screenshot shows a dialog box titled "Adapter Configuration Wizard - Step 4 of 7". The main heading is "Operation". Below the heading, there is a paragraph of text: "The File Adapter supports four operations. There is a Read File operation that polls for incoming files in your local file system, a Write File operation that creates outgoing files, a Synchronous Read File operation that reads the current contents of a file, and a List Files operation that lists file names in specified locations. Specify the Operation type and Operation Name. Only one operation per Adapter Service may be defined using this wizard." Below this text, there are four radio button options under the label "Operation Type": "Read File", "Write File" (which is selected and highlighted with a yellow box), "Synchronous Read File", and "List Files". Below the radio buttons, there is a text input field labeled "Operation Name:" with the word "Write" entered. At the bottom of the dialog, there are four buttons: "Help", "< Back", "Next >" (which is highlighted in blue), "Finish", and "Cancel".

6. 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

Adapter Configuration Wizard - Step 5 of 7

File Configuration

Specify the parameters for the Write File operation.

Directory specified as Physical Path Logical Name

Directory for Outgoing Files (physical path):
OutputDir

File Naming Convention (po_%SEQ%.txt): PO_%SEQ%.xml

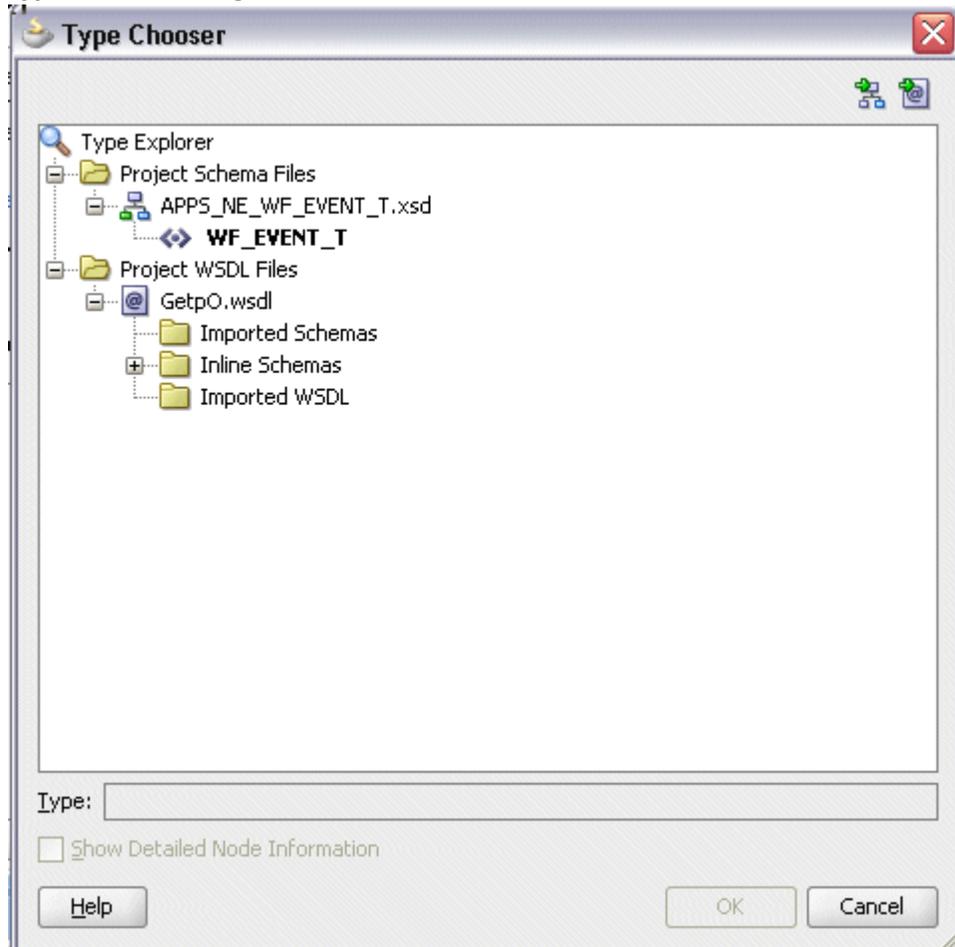
Append to existing file

Write to output file when any of these conditions are met:

<input checked="" type="checkbox"/> Number of Messages Equals:	1	
<input type="checkbox"/> Elapsed Time Exceeds:	1	minutes
<input type="checkbox"/> File Size Exceeds:	1000	kilobytes

7. 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.
8. Confirm the default write condition: Number of Messages Equals 1. Click **Next**. The Messages dialog box appears.
9. Select **Browse for schema file** in front of the URL field.
The Type Chooser dialog is displayed.

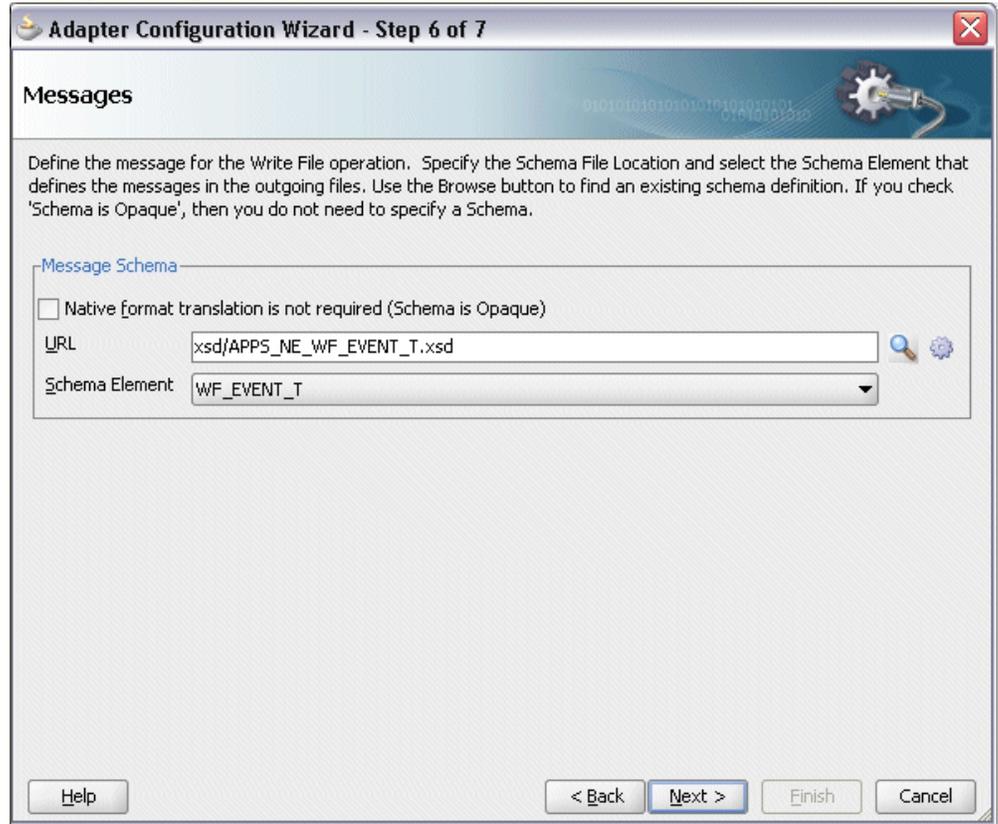
Type Chooser Dialog



Expand the Project Schema Files and select WF_EVENT_T schema from the APPS_NE_WF_EVENT_T.xsd. Click **OK**.

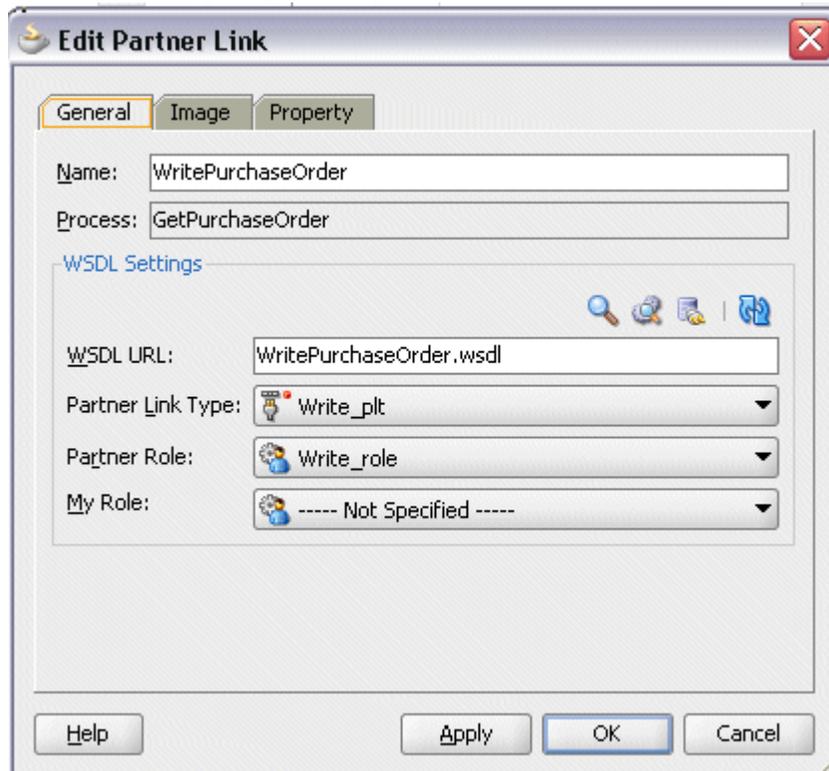
The selected APPS_NE_WF_EVENT_T.xsd is displayed as URL and the WF_EVENT_T element is selected as Schema Element.

Messages Dialog with Selected Schema



10. 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`.

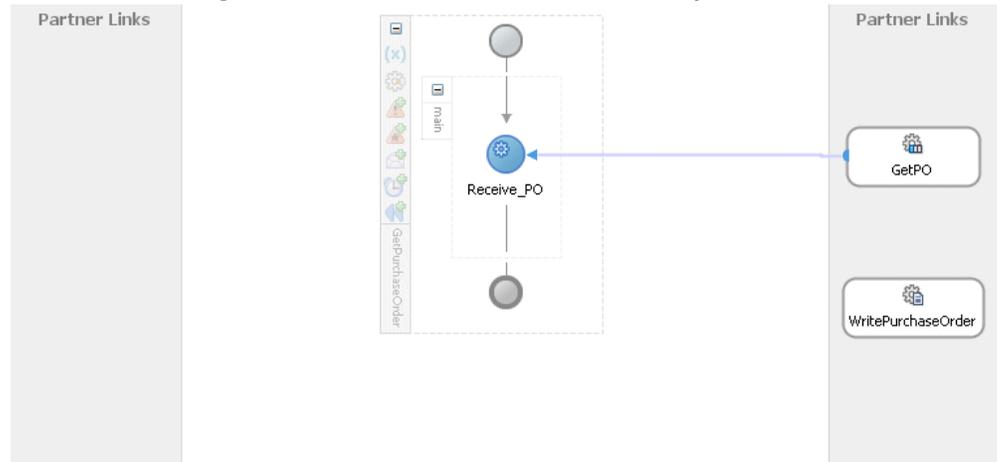
Edit Partner Link Dialog



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 `WritePurchaseOrder` Partner Link appears in the BPEL process diagram.

BPEL Process Diagram with Partner Link Added for File Adapter



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.

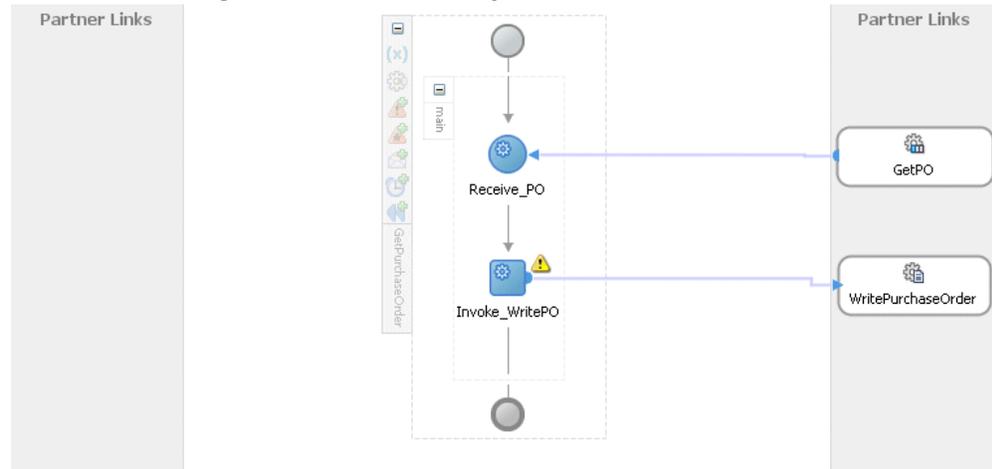
To add an Invoke activity:

1. In Oracle JDeveloper, expand the **BPEL Constructs** from the Component Palette. Drag and drop the **Invoke** activity from the Component Palette into the center swim lane 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.
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.

BPEL Process Diagram with Invoke Activity Added



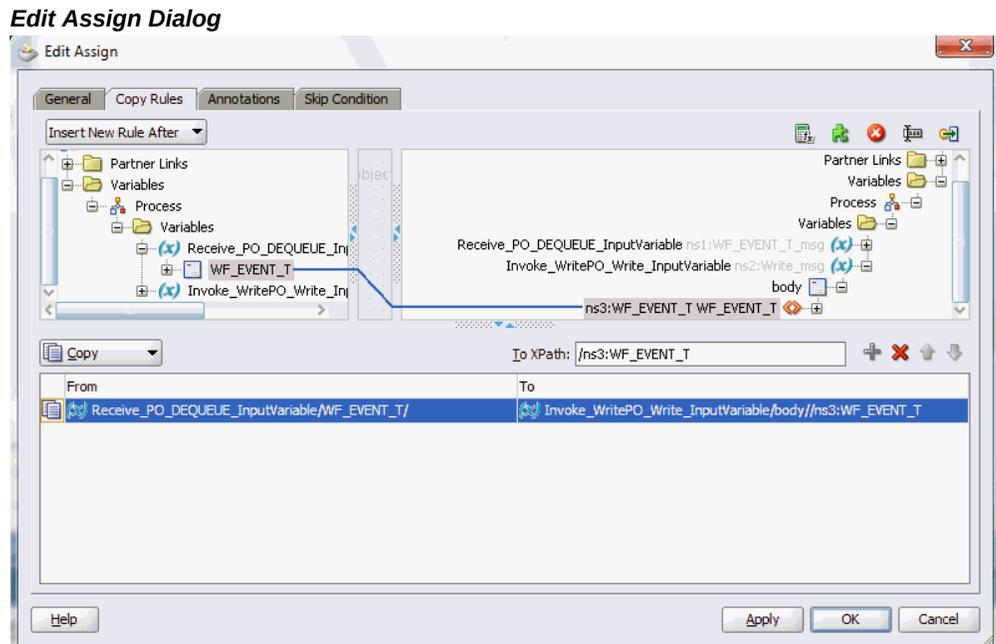
Adding an Assign Activity

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 Oracle JDeveloper, expand the **BPEL Constructs** from the Component Palette. Drag and drop the **Assign** activity from the Component Palette into the center swim lane of the process diagram, between the **Receive** activity and the **Invoke** activities.
2. Double-click the **Assign** activity to access the Edit Assign dialog box.
3. Select the Copy Rules tab and expand the source and target trees:
 - In the From navigation tree, navigate to **Variable > Process > Variables > Receive_PO_Dequeue_InputVariable** and select **WF_EVENT_T** element.
 - In the To navigation tree, navigate to **Variable > Process > Variables > Invoke_WritePO_Write_InputVariable** and select **body**.

Drag the source node (**WF_EVENT_T**) to connect to the target node (**body**) that you just identified. This creates a line that connects the source and target nodes. The copy rule is displayed in the From and To sections at the bottom of the Edit Assign dialog box.



4. Click **Apply** and then **OK** in the Edit Assign dialog box to complete the configuration of the Assign activity.

Click the `composite.xml` to display the Oracle JDeveloper composite diagram.

Deploying and Testing the SOA Composite Application with BPEL Process at Runtime

After creating a SOA composite application with the subscribed event in BPEL process, you need to deploy it to the Oracle WebLogic managed server. This can be achieved using Oracle JDeveloper. Once the composite is deployed, it can be tested from the Oracle Enterprise Manager Fusion Middleware Control Console.

Prerequisites

Before deploying the SOA composite with BPEL process using Oracle JDeveloper, you must have established the connectivity between the design-time environment and the runtime server. For information on how to configure the necessary server connection, see *Configuring Server Connection*, page B-1.

Note: If a local instance of the Oracle WebLogic Server is used, start the WebLogic Server by selecting `Run > Start Server Instance` from Oracle JDeveloper. Once the WebLogic Admin Server "DefaultServer" instance is successfully started, the `<Server started in Running mode>` and

DefaultServer started message in the Running:DefaultServer and Messages logs should appear.

Perform the following runtime tasks:

1. Deploy the SOA Composite Application with BPEL Process, page 6-22
2. Manually initiate the SOA Composite Application with BPEL Process, page 6-23

Deploying the SOA Composite Application with BPEL Process

Before manually testing the BPEL process, you need to deploy it first.

To deploy the SOA composite application with BPEL process:

1. In the Applications Navigator of JDeveloper, select the **GetPOProject** project.
2. Right-click the project and select **Deploy > [project name] > [serverConnection]** from the menu.

For example, you can select **Deploy > GetPOProject > SOAServer** to deploy the process if you have the connection appropriately.

Note: If this is the first time to set up the server connection, then the Deployment Action dialog appears. Select 'Deploy to Application Server' and click **Next**.

In the Deploy Configuration dialog, ensure the following information is selected before clicking **Next** to add a new application server:

- New Revision ID: 1.0
- Mark composite revision as default: Select this checkbox.
- Overwrite any existing composites with the same revision ID: Select this checkbox.

The steps to create a new Oracle WebLogic Server connection from Oracle JDeveloper are covered in Configuring Server Connection, page B-1.

3. In the Select Server dialog, select 'soa-server1' that you have established the server connection earlier.

Click **Next**.

4. In the SOA Servers dialog, accept the default target SOA Server ('soa-server1')

selection.

Click **Next** and **Finish**.

5. If you are deploying the composite for the first time from your Oracle JDeveloper session, the Authorization Request window appears. Enter username and password information specified during Oracle SOA Suite installation. Click **OK**.
6. Deployment processing starts. Monitor deployment process and check for successful compilation in the SOA - Log window.
Verify that the deployment is successful in the Deployment - Log window.

Testing the SOA Composite Application with BPEL Process

Once the BPEL process contained in the SOA composite application has been successfully deployed, you can manage and monitor the process from Oracle Enterprise Manager Fusion Middleware Control Console. You can also test the process and the integration interface by manually initiating the process.

Log in 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 contained in the SOA composite application:

1. Navigate to Oracle Enterprise Manager Fusion Middleware Control Console (<http://<hostname>:<port>/em>). The login page appears.
2. Enter the username and password information specified during the Oracle SOA Suite installation. Click **Login** to log in to a farm. The composite (GetPurchaseOrder) you deployed is displayed in the Applications Navigation tree.

You may need to select an appropriate target instance farm if there are multiple target Oracle Enterprise Manager Fusion Middleware Control Console farms.

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

3. From the Farm navigation pane, expand the SOA > soa-infra node in the tree to navigate through the SOA Infrastructure home page and menu to access your deployed SOA composite applications running on soa-infra managed server.
Ensure that GetPurchaseOrder has been deployed.
4. Log in to Oracle E-Business Suite as a user who has 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: Enter a trading partner name, such as 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

Transaction Type	Transaction SubType	Standard Code	External Transaction Type	External Transaction SubType	Direction Map	Connection/Hub	Protocol Type
PO	PRO	OAG	PO	PROCESS	OUT	itg_process_p	SOAP

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 Orders form:
 - 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

Purchase Orders - 5789

Operating Unit: Vision Operations | Created: 07-JUN-2008 01:29:02

PO, Rev: 5789 | 0 | Type: Standard Purchase Order

Supplier: | Site: | P-Card: | Contact: |

Ship-To: M1- Seattle Mfg | Bill-To: V1- New York City | Currency: USD

Buyer: | Status: Approved | Total: 150.39

Description: |

Lines | Price Reference | Reference Documents | More | Agreement | Temporary Labor

Num	Type	Item	Rev	Job	Category	Description	UOM	Quantity	Price
1	Goods	CM13139			PRODUCTN.DRN	Hard Drive - 8GB	Each	1	150.393

Item: CM13139 | Hard Drive - 8GB

Catalog... | Currency... | Terms | Shipments | Approve...

The status of the purchase order is now changed to 'Approved'. For future reference, record 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 Enterprise Manager Fusion Middleware Control Console to confirm that the `GetPurchaseOrder` BPEL process has been completed.
To verify, click the Instances tab. The SOA composite application instance ID, name, conversation ID, most recent known state of each instance since the last data refresh of the page are displayed.
14. Click your BPEL service component instance link (such as `GetPurchaseOrder`) to display the Instances page where you can view the process details of the BPEL activities in the Audit Trail tab. Click the Flow tab to check the BPEL process flow diagram.
15. Double-click on 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.

Verification of the Receive Event Name

```
<?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>
  </PARAMETER_LIST_ITEM>
</PARAMETER_LIST>
</WF_EVENT_T>
</xml>
```

16. Examine the Assign and Invoke activities as well for the event raised and document number.
17. 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.

Confirmation of the Output 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>3768</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_PARAMETERS3</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_PARAMETERS5</NAME>
    <VALUE>204</VALUE>
  </PARAMETER_LIST_ITEM>
  <PARAMETER_LIST_ITEM>
    <NAME>Ecx_PARTY_ID</NAME>
```

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.

Concurrent programs can be exposed as SOAP-based and REST-based services. This chapter takes a concurrent program SOAP service as an example to explain the service invocation. A deployed SOAP service can be orchestrated into a meaningful BPEL process within a SOA composite application with service endpoints. At runtime, the SOA Composite in the WebLogic managed server where the soa-infra application is running can be exposed to customers and invoked through any of the web service clients or orchestration tools including Oracle JDeveloper, Apache Axis, .NET Web Service Client, Oracle BPEL Process Manager, and Oracle Enterprise Service Bus (ESB).

This chapter discusses how to create a SOA composite application with BPEL process to invoke the web service and how to use it to update Oracle E-Business Suite. For the example described in the following sections, Oracle JDeveloper 11g (11.1.1.6.0) is used as a design-time tool to create a SOA composite application and Oracle SOA Suite 11g (11.1.1.6.0) is used for the process deployment.

Note: While using Oracle JDeveloper with other Oracle Fusion Middleware components (such as Oracle SOA Suite), to enable SOA technologies, you need to manually download Oracle SOA Suite Composite Editor, an Oracle JDeveloper extension for SOA technologies. For more information on installing additional Oracle Fusion Middleware design-time components, see the *Oracle Fusion Middleware Installation Guide for Oracle JDeveloper*.

Using Concurrent Program WSDLs at Design Time

SOA Composite Application with 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 contained in a SOA composite application, a sales order information including header and line items is read by a File Adapter. The sales order data will then be passed to create a departure shipment notice (DSNO). The shipment notice creation document number will be passed back to the request application.

When the BPEL process has been successfully invoked after the deployment, you can validate the process to see if the generated shipment notice has correct trading partner information as described in the sales order.

Prerequisites to Create a SOA Composite Application with BPEL Process Scenario 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 administrator or an integration developer needs to generate a web service first. The administrator will deploy the generated service to an Oracle SOA Suite WebLogic managed server.
- An integration developer needs to locate and record the deployed WSDL URL for the concurrent program exposed as a web service.
- SOAHeader elements should be populated in order to run the concurrent program for the SOAP request

Deploying a Concurrent Program Web Service Composite

Once a web service for the selected interface definition has been generated successfully, the administrator then can deploy the service from Oracle Integration Repository to an Oracle SOA Suite WebLogic managed server.

For example, the following steps must be performed first before the integration developer can create a SOA composite application with BPEL process using a deployed WSDL:

1. To generate a web service, the integration administrator or the integration developer locates the interface definition first from the Oracle Integration Repository (such as Departure Shipment Notice Outbound WSHDSNO concurrent program) and clicks **Generate** in the SOAP Web Service tab of the interface details page.

For detailed instructions 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, the administrator selects one authentication type before clicking **Deploy** in the SOAP Web Service tab of the interface details page. The deployed service in Oracle SOA Suite is an active service and is ready to accept new SOAP requests.

Once the service has been successfully deployed, the selected authentication type will be displayed along with 'Deployed' with 'Active' state in the SOAP Service Status field. 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 information 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 a WSDL URL

Apart from the required tasks performed by the administrator, the integration developer needs to locate and record the deployed web service WSDL URL for the interface (such as WSHDSNO concurrent program) that needs to be orchestrated into a meaningful BPEL process in Oracle JDeveloper.

This WSDL information will be used directly for a partner link during the BPEL process creation at design time.

Concurrent Program Interface Details Page

Integration Repository >

Concurrent Program : Departure Shipment Notice Outbound [Browse](#) [Search](#) [Printable Page](#)

Internal Name: WSHDSNO Scope: Public
 Type: Concurrent Program Interface Source: Oracle
 Product: Shipping Execution Common
 Status: Active
 Business Entity: [Trip](#)

Overview | **SOAP Web Service** | REST Web Service

SOAP Service Status: Deployed | Active | [View WSDL](#) Log Configuration: Disabled

Service Operations

Expand All | Collapse All

Display Name	Internal Name	Grant
Departure Shipment Notice Outbound	WSHDSNO	
Process	Process	

Web Service Security

* Authentication Type: Username Token SAML Token (Sender Vouches)

Copyright (c) 1998, 2020, Oracle and/or its affiliates. All rights reserved. [About this Page](#) [Privacy Statement](#)

For information on how to search for an interface and review the interface details, see *Searching and Viewing Integration Interfaces*, page 2-1.

Setting Variables in SOAHeader for a SOAP Request

Certain variables required to set application context must be populated for the SOAHeader elements to pass values during the service invocation. These SOAHeader elements for concurrent program interface type are *Responsibility*, *RespApplication*, *SecurityGroup*, *NLSLanguage*, and *Org_Id*.

Note: The user name and password information is defined by the web service security policy (such as `oracle/wss_username_token_service_policy`). Detailed instructions on how to pass the security headers along with the SOAP request, see *Configuring Web Service Policies*, page 7-19.

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 invocation 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.

SOA Composite Application with BPEL Process Creation Flow

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

1. Create a New SOA Composite Application with BPEL Process, page 7-6
Use this step to create a new SOA composite application with BPEL process 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 to the request application.
2. Create a Partner Link, page 7-7
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 synchronously read sales order details received from the trading partner.

4. Add Invoke Activities, page 7-13

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. Pass the SOAHeader variables for the invocation of the DSNO concurrent program service.
2. Pass the order details from the output of the Synchronous Read - File Adapter service to the input of the DSNO creation.
3. Set the SOAP response to output.

For general information and how to create SOA composite applications using BPEL process service component, see the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite* for details.

Creating a New SOA Composite Application with BPEL Process

Use this step to create a new SOA composite application that will contain various BPEL process activities.

To create a new SOA composite application with BPEL process:

1. Open Oracle JDeveloper.
2. Click **New Application** in the Application Navigator.
The "Create SOA Application - Name your application" page is displayed.
3. Enter an appropriate name for the application in the Application Name field and select **SOA Application** from the Application Template list.
Click **Next**. The "Create SOA Application - Name your project" page is displayed.
4. Enter an appropriate name for the project in the Project Name field, for example, ShipNotice.
5. In the Project Technologies tab, select 'Web Services' and ensure that **SOA** is

selected from the Available technology list to the Selected technology list.

Click **Next**. The "Create SOA Application - Configure SOA settings" page is displayed.

6. Select **Composite With BPEL Process** from the Composite Template list, and then click **Finish**. You have created a new application, and a SOA project. This automatically creates a SOA composite.

The Create BPEL Process page is displayed.

7. Leave the default **BPEL 1.1 Specification** selection unchanged. This creates a BPEL project that supports the BPEL 1.1 specification.

Enter an appropriate name for the BPEL process in the Name field, for example `ShipNotice`.

Select **Synchronous BPEL Process** in the Template field.

Select **required** from the Transaction drop-down list. Click **OK**.

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

8. Navigate to SOA Content > Business Rules and double click `composite.xml` to view the composite diagram.

Double click on the `ShipNotice` component to open the BPEL process.

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 the `WSHDSNO` concurrent program.

To create a partner link for `Shipment_Notice` web service:

1. In Oracle JDeveloper, place your mouse in the Partner Links area and right click on mouse to select **Create Partner Link...** from the pull-down menu. Alternatively, you can drag and drop **Partner Link** from the **BPEL Constructs** list into the right Partner Link swim lane of the process diagram.

The Create Partner Link window appears.

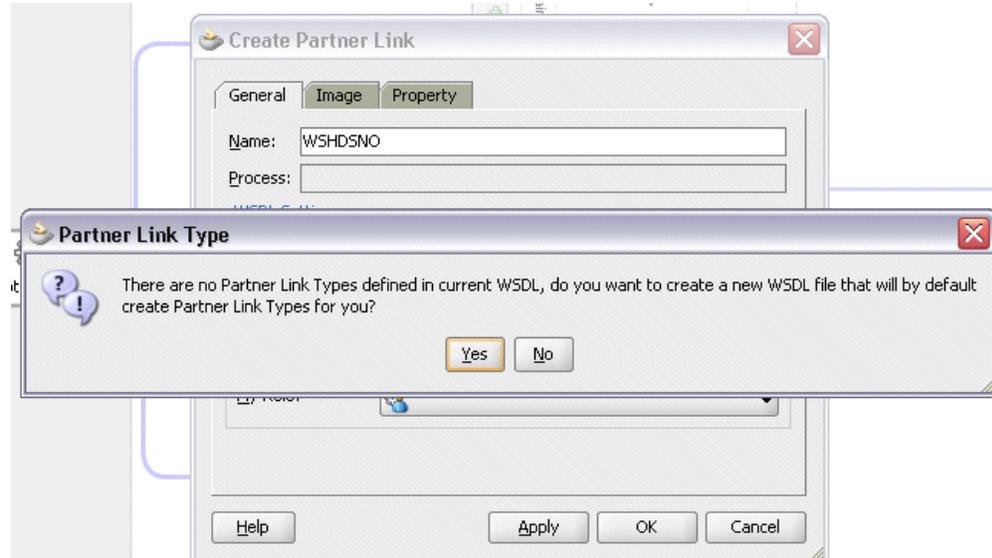
2. Copy the WSDL URL corresponding to the Departure Shipment Notice Outbound `WSHDSNO` service that you recorded earlier in the WSDL File field.

Press the **[Tab]** key.

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.

Partner Link Type Message Dialog



Click **Yes** to have the Partner Name value populated automatically. You can manually enter the Name such as WSHDSNO.

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. 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 Link Type and Partner Role fields from the drop-down lists. Click **Apply**.

5. 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 Oracle JDeveloper, drag and drop the **File Adapter** service from the **BPEL Services** list into the right Partner Link swim lane 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.
4. Click **Next**. The Adapter Interface dialog box appears.
5. Select the **Define from operation and schema (specified later)** radio button and click **Next**. The Operation dialog box appears.

Operation Dialog

Operation

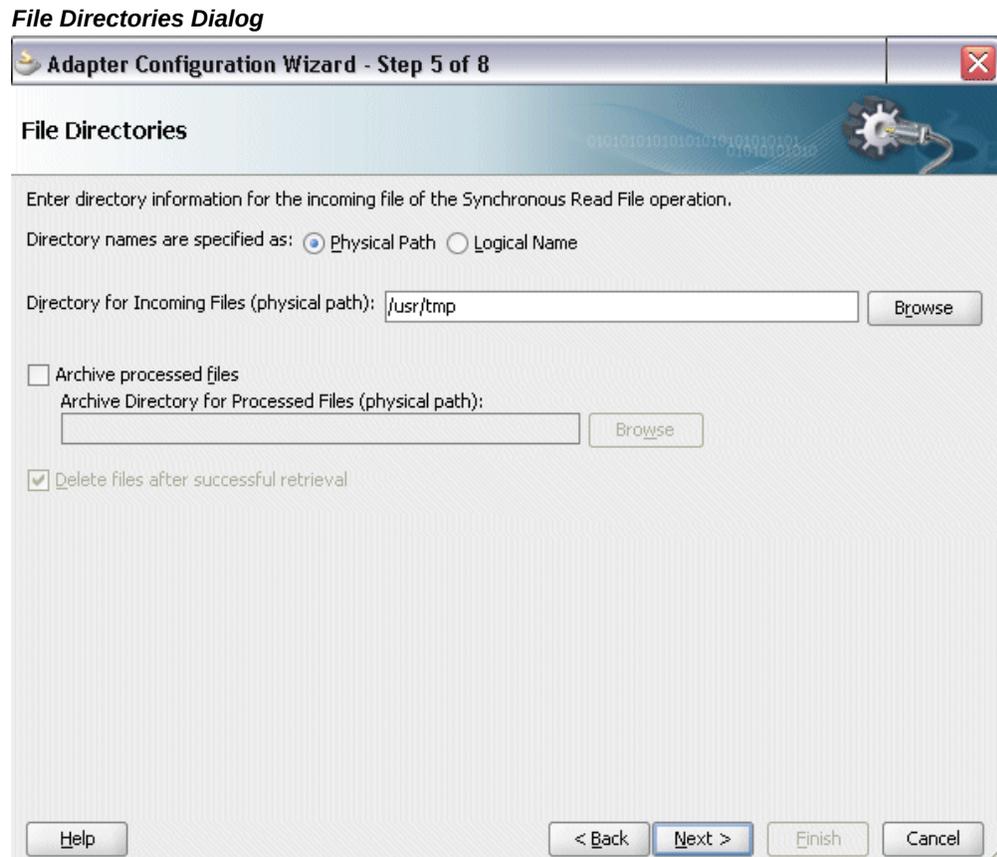
The File Adapter supports four operations. There is a Read File operation that polls for incoming files in your local file system, a Write File operation that creates outgoing files, a Synchronous Read File operation that reads the current contents of a file, and a List Files operation that lists file names in specified locations. Specify the Operation type and Operation Name. Only one operation per Adapter Service may be defined using this wizard.

Operation Type: Read File
 Write File
 Synchronous Read File
 List Files

Operation Name:

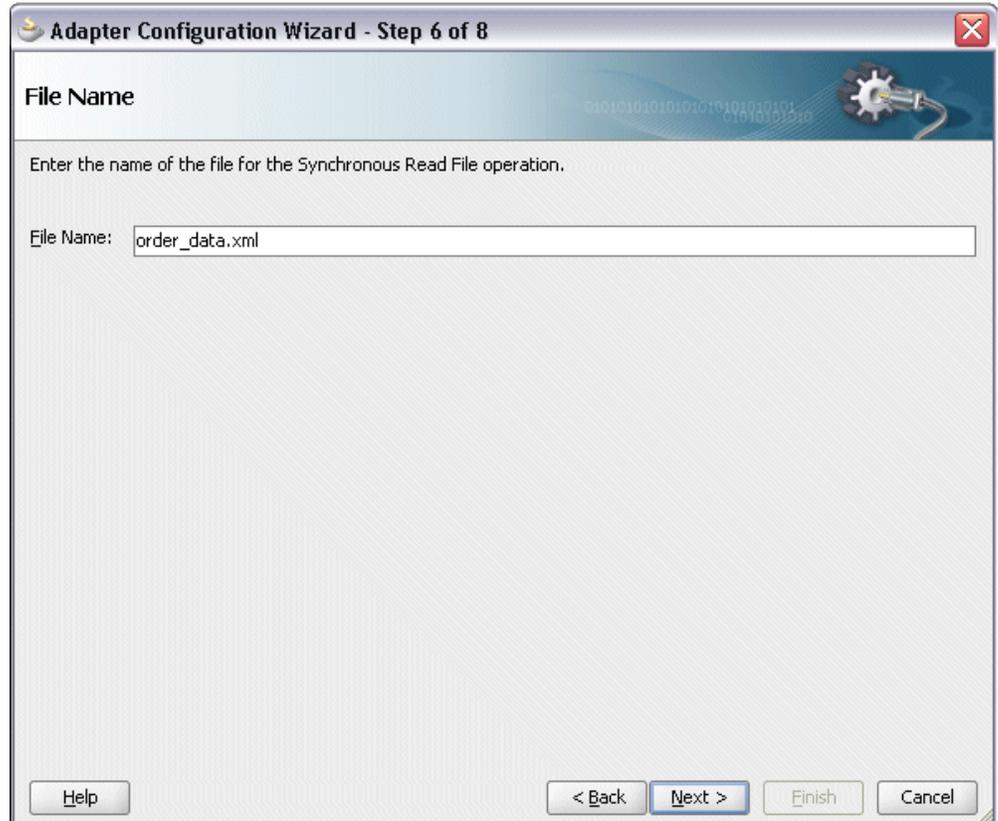
Help < Back Next > Finish Cancel

6. 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.



7. Select the **Physical Path** radio button and enter the input payload file directory information. For example, enter `/usr/tmp/` as the directory name.
Click **Next** to open the File Name dialog box.
8. Enter the name of the file for the synchronous read file operation, for example `'order_data.xml'`.

File Name Dialog



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

9. Select **Browse for schema file** in front of the URL field. The Type Chooser window is displayed.
 1. Click the **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://<soa_suite_hostname>:<port>/soa-infra/services/default/<jndi_name>_CONCURRENTPROGRAM_WSHDSNO/WSHDSNO_Service?XSD=APPS_ISG_CP_REQUEST_CP_SUBMIT.xsd`.

Schema location for your service can be found from the service WSDL URL (for example, `http://<soa_suite_hostname>:<port>/soa-infra/services/default/<jndi_name>_CONCURRENTPROGRAM_WSHDSNO/WSHDSNO_Service?wsdl`).
 3. Select the **Copy to Project** checkbox and click **OK**.

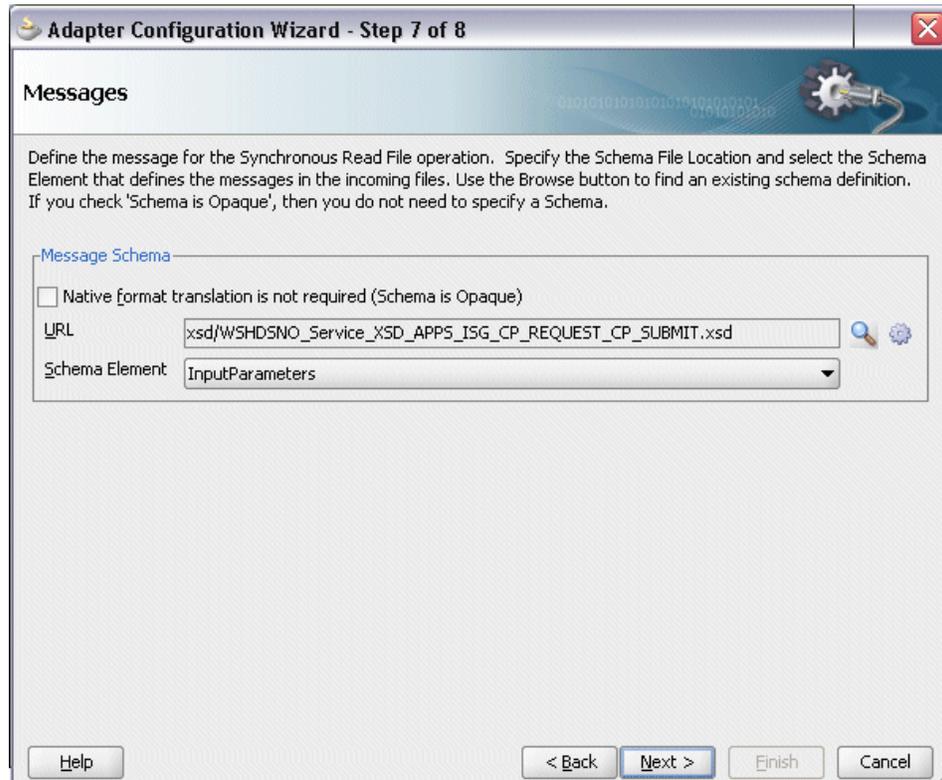
The Localize Files window appears. Ensure the **Maintain original directory**

structure for imported files checkbox is selected and click **OK**.

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

4. Expand the Imported Schema folder and select InputParameters Message in the WSHDSNO_Service_XSD_APPS_ISG_CP_REQUEST_CP_SUBMIT.xsd. Click **OK**.
5. The selected .xsd is displayed as URL, and the InputParameters is selected as Schema Element.

Messages Dialog with Selected Message Schema



10. 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.

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.

Adding Invoke Activities

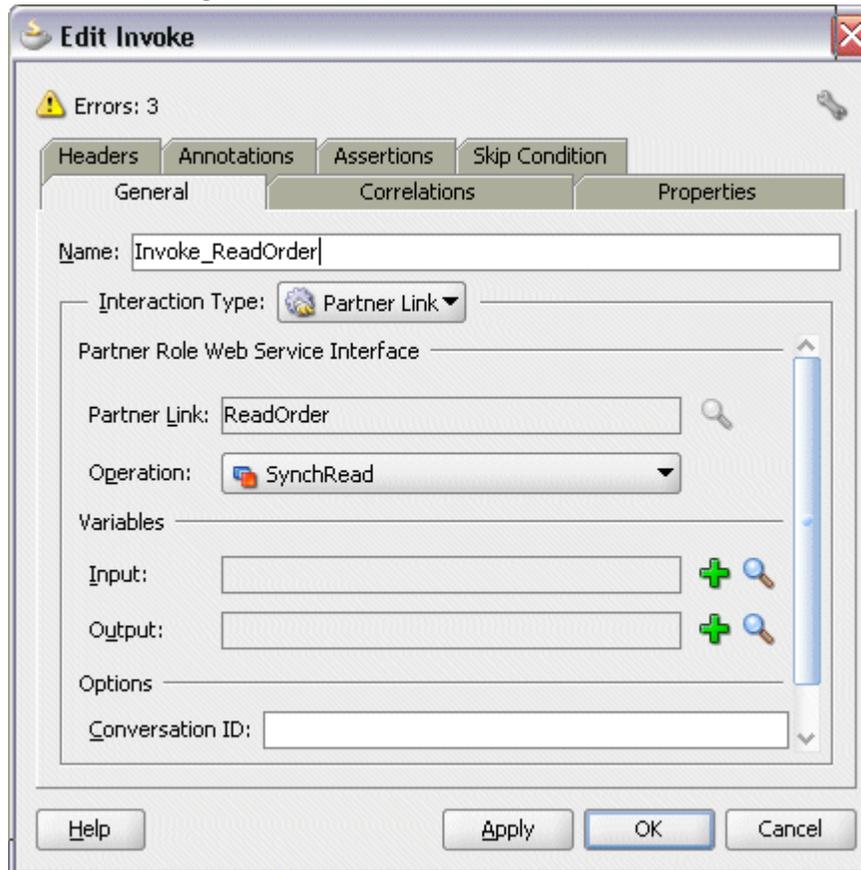
This step is to configure two Invoke activities in order to:

- Read order details 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 and generate an outbound shipment notice (DSNO) through the `Shipment_Notice` partner link.

To add an Invoke activity for ReadOrder Partner Link:

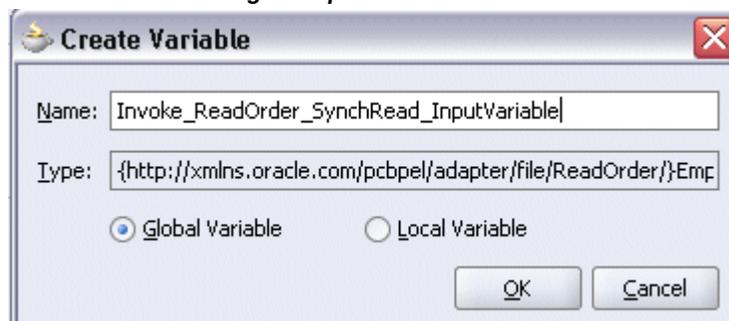
1. In Oracle JDeveloper, expand the **BPEL Constructs** from the Component Palette. Drag and drop the **Invoke** activity from the Component Palette into the center swim lane of the process diagram, between the **receiveInput** and **replyOutput** activities.
2. Link the Invoke activity to the `ReadOrder` 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 for Input Variable



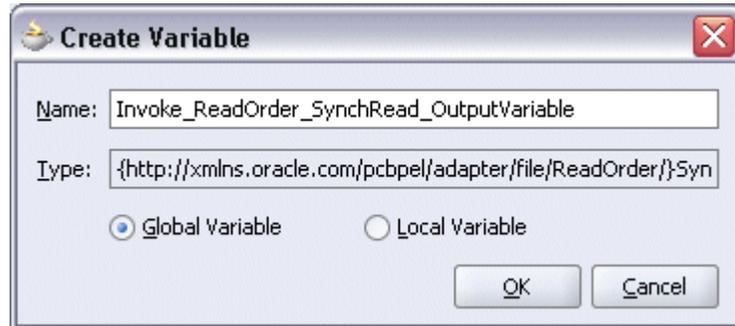
4. Enter a name for the variable. You can also accept the default name. Ensure the

Global Variable radio button is selected and click **OK**.

5. Click the **Create** icon next to the **Output Variable** field.

Enter a name for the output variable. You can also accept the default name. Ensure the **Global Variable** radio button is selected and click **OK**.

Create Variable Dialog for Output Variable



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, expand the **BPEL Constructs** from the Component Palette. Drag and drop the second **Invoke** activity from the Component Palette into the center swim lane of the process diagram, after the **Invoke** and **replyOutput** 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.

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.

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. In Oracle JDeveloper, expand the **BPEL Constructs** from the Component Palette. Drag and drop the **Assign** activity into the center swim lane of the process diagram between the two **Invoke** activities you just created earlier.
2. Double-click the **Assign** activity to access the Edit Assign dialog box.
3. Enter 'SOAHeader' as the Assign name in the Edit Assign dialog box. Click **OK**.
4. Select the Copy Rules tab and expand the target trees:
 - Click the Expression icon to invoke the Expression Builder dialog. Enter 'ORDER_MGMT_SUPER_USER' in the Expression box. Click **OK**. The Expression icon with the expression value ('ORDER_MGMT_SUPER_USER') appears in the center of the Edit Assign dialog, between the From and To navigation tree nodes.
 - In the To navigation tree, navigate to **Variables > Process > Variables > Invoke_Shipment_Notice_WSHDSNO_InputVariable >header > ns2: SOAHeader** and select **ns2:Responsibility**.

Drag the Expression icon to connect to the target node (ns2:Responsibility) that you just identified. This creates a line that connects the source and target nodes. The copy rule is displayed in the From and To sections at the bottom of the Edit Assign dialog box.

5. Enter the second pair of parameters by clicking the Expression icon to invoke the Expression Builder dialog.
 - Enter 'ONT' in the Expression box. Click **OK**. The Expression icon with the expression value ('ONT') appears in the center of the Edit Assign dialog, between the From and To navigation tree nodes.
 - In the To navigation tree, navigate to **Variables > Process > Variables > Invoke_Shipment_Notice_WSHDSNO_InputVariable >header > ns2: SOAHeader** and select **ns2:RespApplication**.

Drag the Expression icon to connect to the target node (ns2:RespApplication) that you just identified. This creates a line that connects the source and target nodes. The copy rule is displayed in the From and To sections at the bottom of the Edit Assign dialog box.

6. Enter the third pair of parameters by clicking the Expression icon to invoke the Expression Builder dialog.
 - Enter 'STANDARD' in the Expression box. Click **OK**. The Expression icon with the expression value ('STANDARD') appears in the center of the Edit Assign dialog, between the From and To navigation tree nodes.
 - In the To navigation tree, navigate to **Variables > Process > Variables > Invoke_Shipment_Notice_WSHDSNO_InputVariable >header > ns2:SOAHeader** and select **ns2:SecurityGroup**.

Drag the Expression icon to connect to the target node (ns2:SecurityGroup) that you just identified. This creates a line that connects the source and target nodes. The copy rule is displayed in the From and To sections at the bottom of the Edit Assign dialog box.

7. Enter the fourth pair of parameters by clicking the Expression icon to invoke the Expression Builder dialog.
 - Enter 'AMERICAN' in the Expression box. Click **OK**. The Expression icon with the expression value ('AMERICAN') appears in the center of the Edit Assign dialog, between the From and To navigation tree nodes.
 - In the To navigation tree, select type Variable. Navigate to **Variables > Process > Variables > Invoke_Shipment_Notice_WSHDSNO_InputVariable >header > ns2:SOAHeader** and select **ns2:NLSLanguage**.

Drag the Expression icon to connect to the target node (ns2:NLSLanguage) that you just identified. This creates a line that connects the source and target nodes. The copy rule is displayed in the From and To sections at the bottom of the Edit Assign dialog box.

8. Enter the fifth pair of parameters by clicking the Expression icon to invoke the Expression Builder dialog.
 - Enter '202' in the Expression box. Click **OK**. The Expression icon with the expression value ('202') appears in the center of the Edit Assign dialog, between the From and To navigation tree nodes.
 - In the To navigation tree, select type Variable. Navigate to **Variables > Process > Variables > Invoke_Shipment_Notice_WSHDSNO_InputVariable >header > ns2:SOAHeader** and select **ns2:Org_Id**.

Drag the Expression icon to connect to the target node (ns2:NLSLanguage) that you just identified. This creates a line that connects the source and target nodes. The copy rule is displayed in the From and To sections at the bottom of the Edit Assign dialog box.

9. 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 Oracle JDeveloper, expand the **BPEL Constructs** from the Component Palette. Drag and drop the **Assign** activity into the center swim lane 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 'SetOrderDetails'.
4. Select the Copy Rules tab and expand the source and target trees:
 - In the From navigation tree, navigate to **Variables > Process > Variables > InvokeReadOrder_SynchRead_OutputVariable > body > InputParameters** and select **ns1:InputParameters**.
 - In the To navigation tree, navigate to **Variables > Process > Variables > InvokeShipmentNotice_WSHDSNO_InputVariable > body** and select **ns1:InputParameters**.

Drag the source node (ns1:InputParameters) to connect to the target node (ns1:InputParameters) that you just identified. This creates a line that connects the source and target nodes. The copy rule is displayed in the From and To sections at the bottom of the Edit Assign dialog box.

5. 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 **BPEL Constructs** section of the Component Palette into the center swim lane of the process diagram, between the **Invoke** and the **Reply** activities.
2. Repeat Step 2 to Step 3 described in creating the first Assign activity to add the third Assign activity called 'SetCPdetails'.
3. Select the Copy Rules tab and expand the target trees:

- In the From navigation tree, navigate to **Variables > Process > Variables > InvokeShipmentNotice_WSHDSNO_OutputVariable** and select **body**.
- In the To navigation tree, navigate to **Variables > Process > Variables > OutputVariable** and select **payload**.

Drag the source node (body) to connect to the target node (payload) that you just identified. This creates a line that connects the source and target nodes. The copy rule is displayed in the From and To sections at the bottom of the Edit Assign dialog box.

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

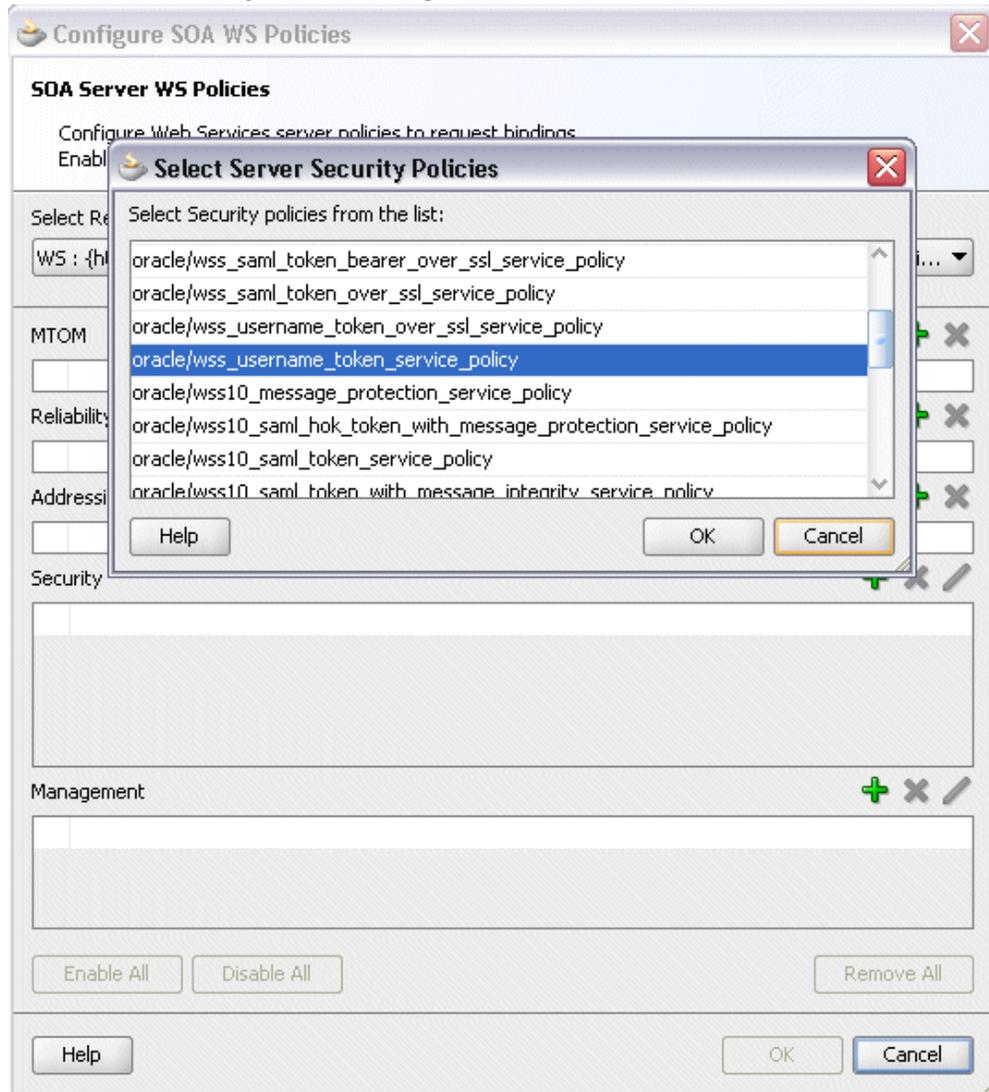
Configuring Web Service Policies

Use the following steps to add a security policy at design time:

1. Navigate to SOA Content > Business Rules > composite.xml. Right click on the `Shipment_Notice` service and select "Configure WS Policies" from the drop-down list.
2. The Configure SOA WS Policies dialog appears.

In the Security section, click the **Add** icon (+). The Select Server Security Policies dialog appears.

Select Server Security Policies Dialog



Select 'oracle/wss_username_token_service_policy' and click **OK**.

The attached security policy is shown in the Security section.

A lock icon appears in the Shipment_Notice service of the composite.xml indicating that a security policy has been successfully attached.

3. From the navigation menu, select **View > Property Inspector** to display the Property Inspector window for Shipment_Notice service component.

In the Properties section, click the **Add** icon (+) for binding properties. The Create Property dialog appears.

Enter 'oracle.webservices.auth.username' in the Name field and enter

'operations' as the value.

Click **OK**.

4. Use the same approach by clicking the **Add** icon (+) again in the Properties section for binding properties. Enter 'oracle.webservices.auth.password' in the Name field. Enter the associated password for user 'operations' in the Value field.

Click **OK**.

Both selected property names and values appear in the Properties section.

Click the Source tab of composite.xml and notice that the oracle.webservices.auth.username and oracle.webservices.auth.password property names and the associated values are added to the Shipment_Notice reference.

Composite.xml File with Property Information

```
36 <reference name="Shipment_Notice" uri:wSDLLocation="WSHDSNO.wsdl">
37 <interface.wSDL interface="http://xmlns.oracle.com/apps/wsh/soapprovider/concurrentprogram/wshdsno/#wsdl.interf
38 <binding.ws port="http://xmlns.oracle.com/apps/wsh/soapprovider/concurrentprogram/wshdsno/#wsdl.endpoint(WSHDSN
39 location="WSHDSNO.wsdl" soapVersion="1.1">
40 <wsp:PolicyReference URI="oracle/wss_username_token_client_policy"
41 orawsp:category="security" orawsp:status="enabled"/>
42 <property name="oracle.webservices.auth.username" type="xs:string"
43 many="false" override="may">operations</property>
44 <property name="oracle.webservices.auth.password" type="xs:string"
45 many="false" override="may">password</property>
46 </binding.ws>
47 </reference>
```

Deploying and Testing the SOA Composite with BPEL Process at Runtime

To invoke the synchronous Departure Shipment Notice Outbound WSHDSNO service from the BPEL client contained in the SOA composite, the SOA composite needs to be deployed to the Oracle WebLogic managed server. This can be achieved using Oracle JDeveloper. Once the composite is deployed, it can be tested from the Oracle Enterprise Manager Fusion Middleware Control Console.

Prerequisites

Before deploying the SOA composite with BPEL process using Oracle JDeveloper, you must have established the connectivity between the design-time environment and the runtime server. For information on how to configure the necessary server connection, see *Configuring Server Connection*, page B-1.

Note: If a local instance of the Oracle WebLogic Server is used, start the WebLogic Server by selecting Run > Start Server Instance from Oracle JDeveloper. Once the WebLogic Admin Server "DefaultServer" instance is successfully started, the <Server started in Running mode> and DefaultServer started message in the Running:DefaultServer and Messages logs should appear.

Perform the following runtime tasks:

1. Deploy the SOA Composite Application with BPEL Process, page 7-22
2. Test the SOA Composite Application with BPEL Process, page 7-23

Deploying the SOA Composite Application with BPEL Process

You must deploy the SOA composite application with BPEL process (`ShipNotice.bpel`) that you created earlier before you can run it.

To deploy the SOA composite application BPEL process:

1. In the Applications Navigator of Oracle JDeveloper, select the **ShipNotice** project.
2. Right-click the project and select **Deploy > [project name] > [serverConnection]** from the menu.

For example, you can select **Deploy > ShipNotice > SOAServer** to deploy the process if you have the connection appropriately.

Note: If this is the first time to set up the server connection, then the Deployment Action dialog appears. Select 'Deploy to Application Server' and click **Next**.

In the Deploy Configuration dialog, ensure the following information is selected before clicking **Next** to add a new application server:

- New Revision ID: 1.0
- Mark composite revision as default: Select this checkbox.
- Overwrite any existing composites with the same revision ID: Select this checkbox.

The steps to create a new Oracle WebLogic Server connection from Oracle JDeveloper are covered in Configuring Server Connection, page B-1.

3. In the Select Server dialog, select 'soa-server1' that you have established the server connection earlier. Click **Next**.
4. In the SOA Servers dialog, accept the default target SOA Server ('soa-server1') selection.
Click **Next** and **Finish**.
5. If you are deploying the composite for the first time from your Oracle JDeveloper

session, the Authorization Request window appears. Enter username and password information specified during Oracle SOA Suite installation. Click **OK**.

6. Deployment processing starts. Monitor deployment process and check for successful compilation in the SOA - Log window.

Verify that the deployment is successful in the Deployment - Log window.

Testing the SOA Composite Application with BPEL Process

Once the BPEL process contained in the SOA composite application has been successfully deployed, you can manage and monitor the process from Oracle Enterprise Manager Fusion Middleware Control Console.

You can also log in 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.

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

To test the SOA composite application with BPEL process:

1. Navigate to Oracle Enterprise Manager Fusion Middleware Control Console (<http://<hostname>:<port>/em>). The login page appears.
2. Enter the username and the password specified during Oracle SOA Suite installation. Click **Login** to log in to a farm. The composite (ShipNotice) you deployed is displayed in the Applications Navigation tree.

You may need to select an appropriate target instance farm if there are multiple target Oracle Enterprise Manager Fusion Middleware Control Console farms.

3. From the Farm navigation pane, expand the SOA >soa-infra node in the tree to navigate through the SOA Infrastructure home page and menu to access your deployed SOA composite applications running on soa-infra managed server.

Click the ShipNotice [1.0] link.

4. Click the Policies tab and notice that the 'oracle/wss_username_token_service_policy' policy you attached to the Shipment_Notice service binding earlier at the design time is now displayed here.
5. In the ShipNotice [1.0] home page, click **Test**.
6. The Test Web Service page for initiating an instance appears. You can specify information as XML payload data to use in the Input Arguments section.

Note: If the WS-Security credentials are not entered at design time, you can enter the credentials at runtime by selecting the WSS Username Token option in the Security section at the top of the Request tab. Enter 'operations' in the Username field and the associated password in the Password field.

Click **Test Web Service** to initiate the process.

The test results appear in the Response tab upon completion.

7. Click your BPEL service component instance link (such as ShipNotice) to display the Instances page where you can view the invocation details of the BPEL activities in the Audit Trail tab.

Click the Flow tab to check the BPEL process flow diagram. Click an activity of the process diagram to view the activity details and flow of the payload through the process.

8. Double-click the Invoke_ShipmentNotice icon from the process flow chart and click the **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 as a user who has 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. Check the Phase and Status of the request to see if the Status of the request is Complete.

Once the concurrent request has been 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 the *Oracle XML Gateway User's Guide* for details.

You can validate if the ship-to address, purchase order, and requested ship date information in the DSNO XML file corresponds with the information in your sales order.

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 process 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 8-4
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 run 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:

Interface Details Page with REST Web Service Tab

Integration Repository Administration

Integration Repository >

Open Interface : AR Autoinvoice Browse Search Printable Page

Internal Name: RAXMTR Status: Active
 Type: Concurrent Program and Open Interface Scope: Public
 Product: Receivables
 Business Entities: [Credit Memo](#), [Debit Memo](#), [Receivables Invoice](#)
 Online Help: [Importing Transaction Information Using AutoInvoice](#), [Oracle Receivables Help](#)

Overview **REST Web Service** Grants

* Service Alias: autoinvoice Log Configuration Disabled **Configure**

REST Service Status: Not Deployed

Service Operations

Name	Direction	Get	Post	Put	Delete	Grant
AR Autoinvoice		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
RA INTERFACE LINES ALL	Inbound	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
RA INTERFACE SALES CREDITS ALL	Inbound	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
RA INTERFACE DISTRIBUTIONS ALL	Inbound	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
RA INTERFACE ERRORS ALL	Inbound	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
SUBMIT_CP_RAXMTR		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

TIP To apply any changes in Operation, Undeploy the service.

REST Service Security

*Authentication Type: HTTP Basic Security Token

Tip: Use [Login Service](#) to obtain Security Token for given user credentials.

Deploy

- 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 checkboxes (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.
 - In the REST Service Security region, ensure that at least one authentication type is selected.
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.

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 '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 checkbox 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 Autoinvoice' 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.

The section includes the following invocation scenarios:

1. Inserting Data to Oracle Receivables Using the HTTP POST Method, page 8-5
2. Updating Data in Oracle Receivables Using the HTTP PUT Method, page 8-7
3. Fetching Data from Oracle Receivables Using the HTTP GET Method, page 8-8
4. Deleting Data in Oracle Receivables Using the HTTP DELETE Method, page 8-9

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.
Sxxxxx</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. Run 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 <username>:<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

Run 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 <username>:<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, run 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 <username>:<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, run the following curl command to delete data from RA_INTERFACE_LINES_ALL with TRX_NUMBER equal to 'Demo1' in 204 ORG_ID:

```
curl -u <username>:<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"
    }
  }
}
```

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 administrator or an integration developer can first generate a SOAP service, and then the administrator can deploy it to an Oracle SOA Suite WebLogic managed server. By leveraging Oracle SOA Suite components, the deployed SOAP service can be orchestrated into a meaningful BPEL process as a SOA composite application with service endpoints. This process can take the data from a business partner and then insert or update Oracle E-Business Suite if necessary.

At runtime, the SOA composite application in the Oracle WebLogic managed server where the `soa-infra` application is running can be exposed to customers and invoked through any of the web service clients or orchestration tools including Oracle JDeveloper, Apache Axis, .NET Web Service Client, Oracle BPEL Process Manager, and Oracle Enterprise Service Bus (ESB).

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, Oracle JDeveloper 11g (11.1.1.6.0) is used as a design-time tool to create the BPEL process and Oracle SOA Suite 11g (11.1.1.6.0) is used for the process deployment.

Note: While using Oracle JDeveloper with other Oracle Fusion

Middleware components (such as Oracle SOA Suite), to enable SOA technologies, you need to manually download Oracle SOA Suite Composite Editor, an Oracle JDeveloper extension for SOA technologies. For more information on installing additional Oracle Fusion Middleware design-time components, see the *Oracle Fusion Middleware Installation Guide for Oracle JDeveloper*.

This chapter includes the following topics:

- Using Business Service Object SOAP Services, page 9-2
 - Using Business Service Object WSDLs at Design Time, page 9-5
 - Deploying and Testing the SOA Composite with BPEL Process at Runtime, page 9-22
- Using Business Service Object REST Services, page 9-25

Using Business Service Object SOAP Services

SOA Composite Application with BPEL Process Scenario

This example uses Purchase Order Service (`/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 information of purchase order details is read by a File Adapter. The order data is then passed to the `approvePurchaseOrder` method within the Purchase Order Service to initiate the single PO approval process. The approval information is then replied back to the requestor.

When the BPEL process has been successfully invoked after the deployment, the purchase order status is changed from `Incomplete` to `Approved`.

Prerequisites to Create a SOA Composite Application with BPEL Process Using a Business Service Object Web Service

- An integration administrator or an integration developer needs to generate a web service first. The administrator will deploy the generated service to an Oracle SOA Suite WebLogic managed 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.

Deploying a Business Service Object Web Service Composite

An integration administrator or an integration developer must first create a SOAP

service for a selected interface definition, and then the administrator can deploy the generated SOAP service from Oracle Integration Repository to an Oracle SOA Suite WebLogic managed server.

For example, the following steps must be performed first before the integration developer can create a BPEL process using the deployed WSDL:

1. To generate a SOAP service, the integration administrator or the integration developer locates the business service object interface definition first from the Oracle Integration Repository (such as Purchase Order Service `/oracle/apps/fnd/framework/toolbox/tutorial/PurchaseOrderService`). In the SOAP Web Service tab of the interface details page, select synchronous interaction pattern for the interface or desired methods from the Interaction Pattern table. Click **Generate** to generate the service.

Once the service has been successfully generated, the Web Service Status field changed from 'Not Generated' to 'Generated'.

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

2. To deploy a generated SOAP service, the administrator selects one authentication type before clicking **Deploy**. The deployed service in Oracle SOA Suite is an active service and is ready to accept new SOAP requests.

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

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

Searching and Recording a WSDL URL

Apart from the required tasks performed by the administrator, the integration developer needs to locate and record the deployed web service WSDL URL for the interface that needs to be orchestrated into a meaningful BPEL process in Oracle JDeveloper.

This WSDL information 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.

Interface Details Page with Deployed WSDL Information

Integration Repository >

Business Service Object : Purchase Order Service

Qualified Name	/oracle/apps/po/service/PurchaseOrderService	Status	Active
Interface	oracle.apps.po.service.PurchaseOrderService	Scope	Public
Extends	oracle.svc.Service	Interface Source	Oracle
Product	Purchasing		

Overview | **SOAP Web Service** | REST Web Service

SOAP Service Status: Deployed | Active | [View WSDL](#) Design Time Log Disabled

Service Operations

Display Name	Internal Name	Synchronous	Grant
▲ Purchase Order Service	/oracle/apps/po/service/PurchaseOrderService	<input checked="" type="checkbox"/>	
create Purchase Order	createPurchaseOrder	<input checked="" type="checkbox"/>	
create Purchase Order Mod	createPurchaseOrderMod	<input checked="" type="checkbox"/>	
update Purchase Order	updatePurchaseOrder	<input checked="" type="checkbox"/>	
update Purchase Order Mod	updatePurchaseOrderMod	<input checked="" type="checkbox"/>	

TIP To apply any changes in Interaction Pattern, Generate or Regenerate the service.

Web Service Security

* Authentication Type: Username Token SAML Token (Sender Vouches)

[Browse](#) [Search](#) [Printable Page](#)

For information on how to search for an interface and review the interface details, see *Searching and Viewing Integration Interfaces*, page 2-1.

Setting Header Variables for a SOAP Request

Certain variables in the BPEL process for header elements must be populated to pass values that may be required to set application context during the service invocation. 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 name and password information is defined by the web service security policy (such as *oracle/wss_username_token_service_policy*). For detailed instructions on how to pass the security headers, along with the SOAP request, see *Configuring Web Service Policies*, page 9-21.

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 invocation 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.

For information on how to set the header variables for the SOAP request, see Assigning ServiceBean_Header Parameters, page 9-16.

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 9-6

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 9-8

This is to synchronously read purchase order details received from the requestor.

4. Add Invoke Activities, page 9-12

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 9-15

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 how to create SOA composite applications using the BPEL process service component, see the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite* for details.

Creating a SOA Composite Application with BPEL Process

Use this step to create a SOA composite application that will contain various BPEL process activities.

To create a new SOA Composite application with BPEL project:

1. Open Oracle JDeveloper BPEL Designer.
2. Click **New Application** in the Application Navigator.
The "Create SOA Application - Name your application" page is displayed.
3. Enter an appropriate name for the application in the Application Name field and

select **SOA Application** from the Application Template list.

Click **Next**. The "Create SOA Application - Name your project" page is displayed.

4. Enter an appropriate name for the project in the Project Name field, for example `ApprovePO`.

5. In the Project Technologies tab, select 'Web Services' and ensure that **SOA** is selected from the Available technology list to the Selected technology list.

Click **Next**. The "Create SOA Application - Configure SOA settings" page is displayed.

6. Select **Composite With BPEL Process** from the Composite Template list, and then click **Finish**. You have created a new application, and a SOA project. This automatically creates a SOA composite.

The Create BPEL Process page is displayed.

7. Leave the default **BPEL 1.1 Specification** selection unchanged. This creates a BPEL project that supports the BPEL 1.1 specification.

Enter an appropriate name (such as `ApprovePO`) for the BPEL process in the Name field.

Select **Synchronous BPEL Process** in the Template field.

Select **required** from the Transaction drop-down list. Click **OK**.

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

8. Navigate to SOA Content > Business Rules and double click `composite.xml` to view the composite diagram.

Double click on the `ApprovePO` component to open the BPEL process.

Creating a Partner Link

Use this step to configure a Partner Link called `PurchaseOrderService`.

To create a partner link for `PurchaseOrderService`:

1. In Oracle JDeveloper, place your mouse in the Partner Links area and right click to select **Create Partner Link...** from the pull-down menu. Alternatively, you can drag and drop **Partner Link** from the **BPEL Constructs** list into the right Partner Link swim lane of the process diagram.

The Create Partner Link window appears.

2. Copy the WSDL URL corresponding to the PurchaseOrderService /oracle/apps/fnd/framework/toolbox/tutorial/PurchaseOrderService that you recorded earlier from the Integration Repository, and paste it in the WSDL File field. Press the **[Tab]** key.
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.

Select the Partner Link Type and Partner Role fields from the drop-down lists. Click **Apply**.
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 synchronously read the purchase order details received from the third party application.

To add a Partner Link for File Adapter:

1. In Oracle JDeveloper, drag and drop the **File Adapter** service from the **BPEL Services** list into the right Partner Link swim lane 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.
4. Click **Next**. The Adapter Interface dialog box appears.
5. Select the **Define from operation and schema (specified later)** radio button and click **Next**. The Operation dialog box appears.
6. 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.

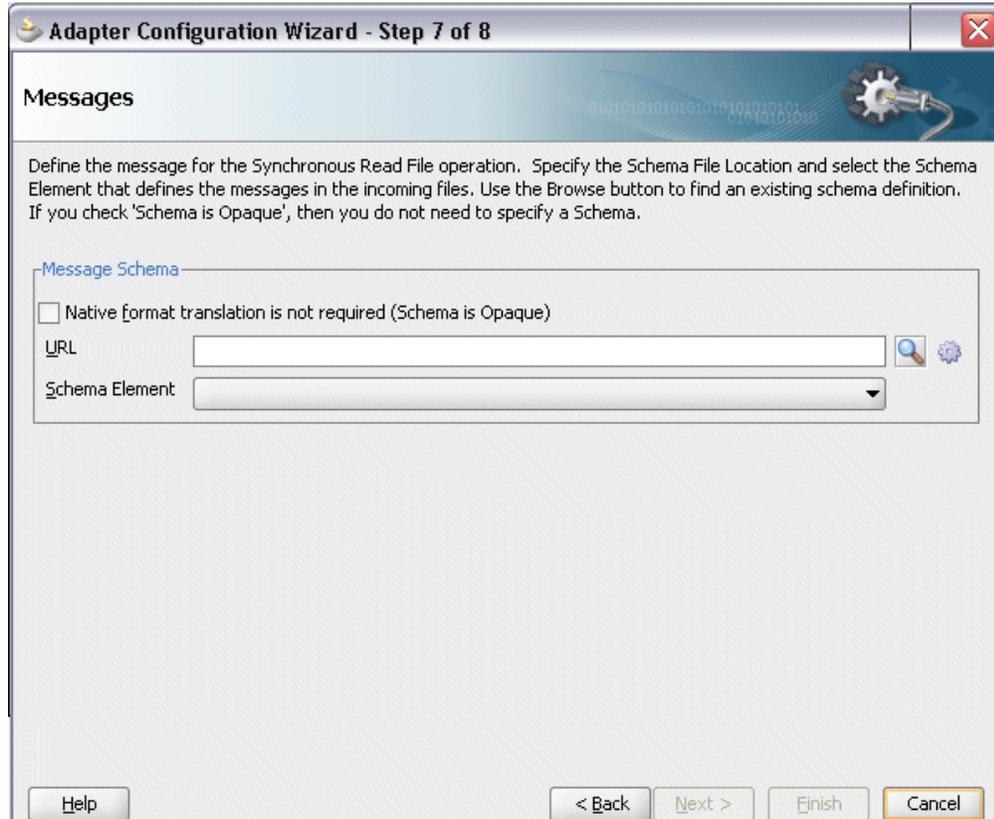
File Directories Dialog

The screenshot shows a dialog box titled "File Directories" within the "Adapter Configuration Wizard - Step 5 of 8". The dialog contains the following elements:

- Header: "File Directories" with a decorative background of binary code and a gear icon.
- Instruction: "Enter directory information for the incoming file of the Synchronous Read File operation."
- Radio buttons: "Directory names are specified as: Physical Path Logical Name".
- Text field: "Directory for Incoming Files (physical path):" with the value "/usr/tmp" and a "Browse" button.
- Checkbox: "Archive processed files" (unchecked).
- Text field: "Archive Directory for Processed Files (physical path):" with an empty field and a "Browse" button.
- Checkbox: "Delete files after successful retrieval" (checked).
- Footer: "Help", "< Back", "Next >", "Finish", and "Cancel" buttons.

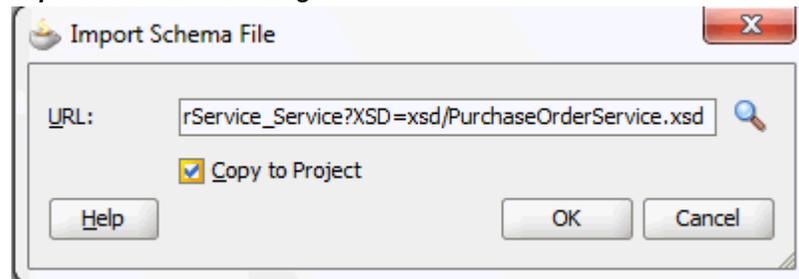
7. Select the **Physical Path** radio button and enter the input payload file directory information. For example, enter /usr/tmp/ as the directory name.
Click **Next** to open the File Name dialog box.
8. Enter the name of the file for the synchronous read file operation. For example, enter 'Input.xml'. Click **Next**. The Messages dialog box appears.

Messages Dialog



9. Select **Browse for schema file** in front of the URL field. The Type Chooser window is displayed.
 1. Click the **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://<soa_suite_hostname>:<port>/soa-infra/services/default/<jndi_name>_PurchaseOrderService/PurchaseOrderService_Service/?XSD=xsd/PurchaseOrderService.xsd`.
- Schema location for your service can be found from the service WSDL URL (for example, `http://<soa_suite_hostname>:<port>/soa-infra/services/default/<jndi_name>_PurchaseOrderService/PurchaseOrderService_Service?wsdl`).

Import Schema File Dialog



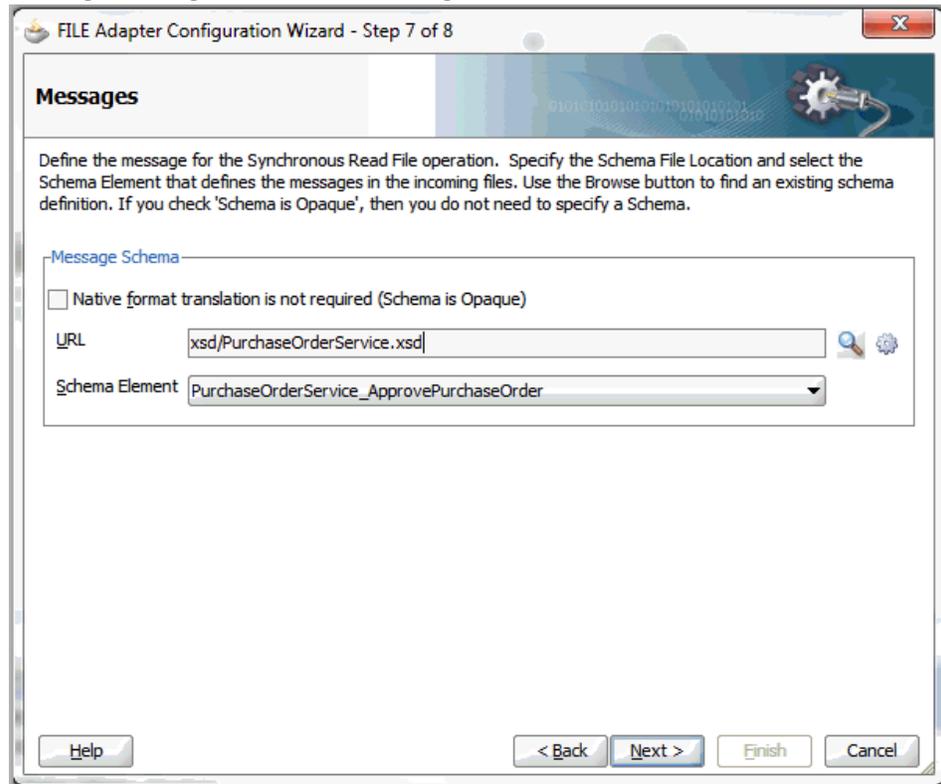
3. Select the **Copy to Project** checkbox and click **OK**.
4. The Localize Files window appears. Ensure the **Maintain original directory structure for imported files** checkbox is selected and click **OK**.

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

5. Expand the Imported Schema folder and select PurchaseOrderService_ApprovePurchaseOrder from the PurchaseOrderService.xsd. Click **OK**.

The selected .xsd is displayed as URL, and the PurchaseOrderService_ApprovePurchaseOrder is selected as Schema Element.

Messages Dialog with Selected Message Schema



10. 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`.

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.

11. Under applications window, navigate to file `ReadPO_file.jca`. Set value of property "DeleteFile" to "false".

Adding an Invoke activity

This step is to configure two Invoke activities to:

1. Point to the File Adapter `ReadPO` to synchronously 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

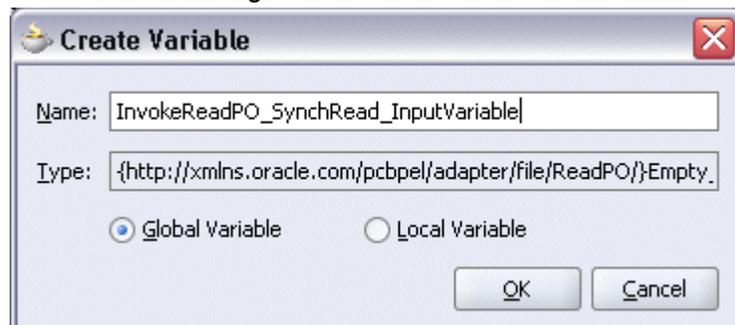
order approval process.

To add an Invoke activity for ReadPO Partner Link:

1. In Oracle JDeveloper, expand the **BPEL Constructs** from the Component Palette. Drag and drop the **Invoke** activity into the center swim lane of the process diagram, between the **receiveInput** and **replyOutput** activities.
2. Link the **Invoke** activity to the ReadPO service. The Edit Invoke dialog box appears.
3. Enter a name for the **Invoke** activity, such as 'InvokeReadPO', and then click the **Create** icon next to the **Input Variable** field to create a new variable. The Create Variable dialog box appears.

Enter an appropriate name in the Name field. You can also accept the default name.

Create Variable Dialog



4. Select **Global Variable** and click **OK**.
5. Click the **Create** icon next to the **Output Variable** field to create a new variable. The Create Variable dialog box appears.

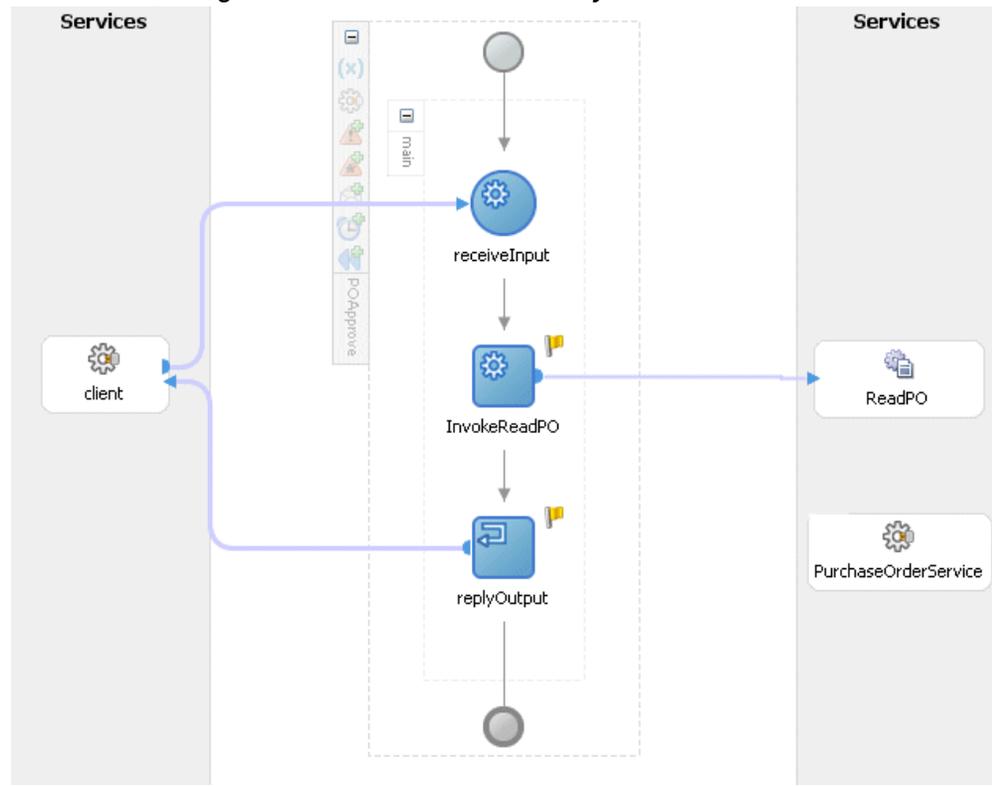
Enter an appropriate name in the Name field. You can also accept the default name.

Select **Global Variable**. 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.

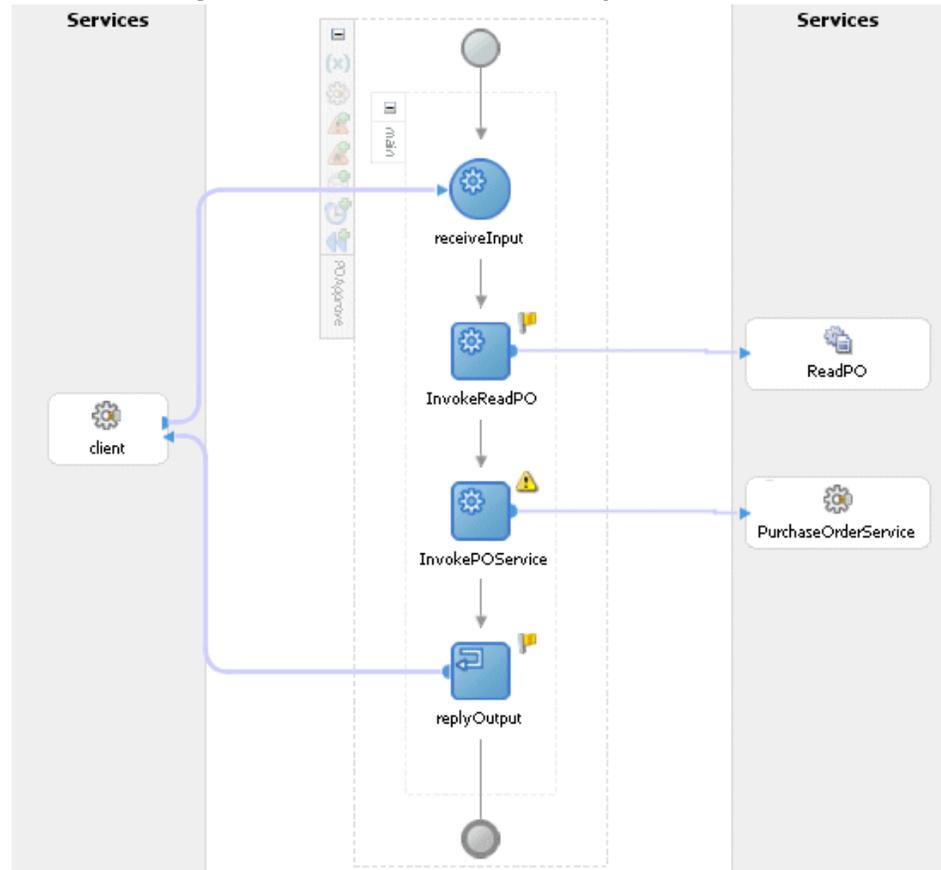
BPEL Process Diagram with the First Invoke Activity Added



To add an Invoke activity for PurchaseOrderService Partner Link:

1. In Oracle JDeveloper, expand the **BPEL Constructs** from the Component Palette. Drag and drop the **Invoke** activity into the center swim lane of the process diagram, after the first **Invoke** activity and the **replyOutput** activity.
2. Link the **Invoke** activity to the **PurchaseOrderService** service. The Edit Invoke dialog box appears.
3. Enter a name for the second **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 second **Invoke** activity.
The second **Invoke** activity appears in the process diagram.

BPEL Process Diagram with the Second Invoke Activity Added



Adding an Assign activity

This step is to configure three Assign activities to:

1. 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 9-16.

2. Pass the purchase order details read from the File Adapter as an input to the second Invoke activity for PurchaseOrderService partner link.
3. Pass single purchase order approval information to the requestor through the dummy Reply activity.

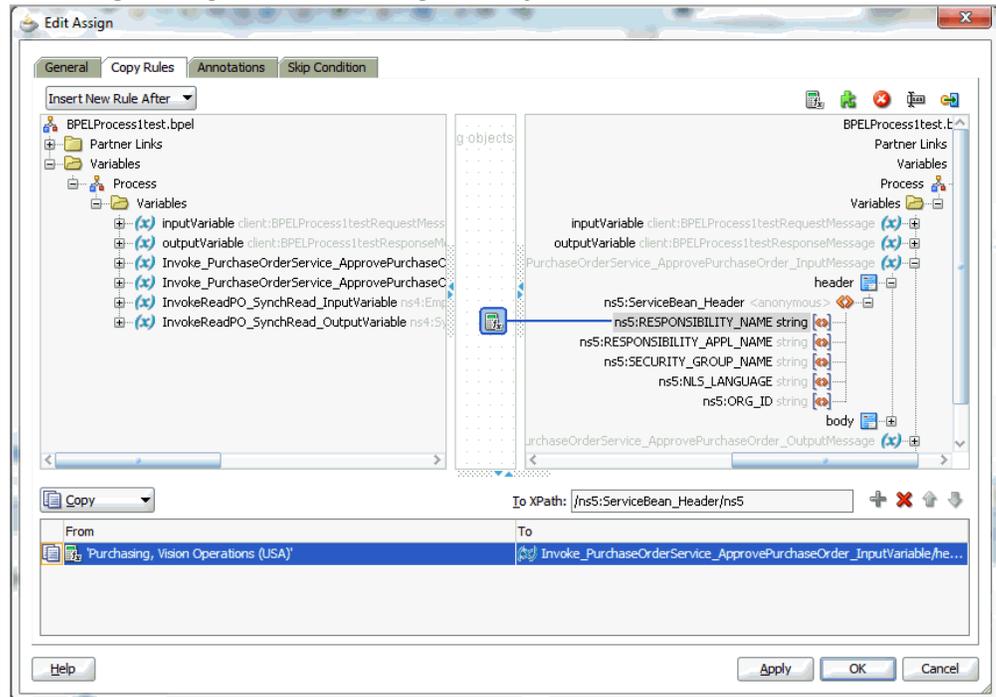
To add the first Assign activity to pass header details:

Assigning ServiceBean_Header Parameters:

1. In Oracle JDeveloper, expand the **BPEL Constructs** from the Component Palette. Drag and drop the **Assign** activity into the center swim lane of the process diagram between the two **Invoke** activities you just created earlier.
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. Select the Copy Rules tab and expand the target trees:
 - Click the Expression icon to invoke the Expression Builder dialog. Enter 'Purchasing, Vision Operations (USA)' in the Expression box. Click **OK**. The Expression icon with the expression value appears in the center of the Edit Assign dialog, between the From and To navigation tree nodes.
 - In the To navigation tree, navigate to **Variables > Process > Variables > InvokePurchaseOrderService_PurchaseOrderService_ApprovePurchaseOrder_InputVariable > header > ns5:ServiceBean_Header** and select **ns5:RESPONSIBILITY_NAME**.

Drag the Expression icon to connect to the target node (ns5:RESPONSIBILITY_NAME) that you just identified. This creates a line that connects the source and target nodes. The copy rule is displayed in the From and To sections at the bottom of the Edit Assign dialog box.

Edit Assign Dialog for the First Assign Activity



5. Enter second pair of parameters by clicking the Expression icon to invoke the Expression Builder dialog.
 - Enter 'PUR' in the Expression box. Click **OK**. The Expression icon with the expression value ('PUR') appears in the center of the Edit Assign dialog, between the From and To navigation tree nodes.
 - In the To navigation tree, navigate to **Variables > Process > Variables > InvokePurchaseOrderService_PurchaseOrderService_ApprovePurchaseOrder_InputVariable > header > ns5:ServiceBean_Header** and select **ns5:RESPONSIBILITY_APPL_NAME**.

Drag the Expression icon to connect to the target node (ns5:RESPONSIBILITY_APPL_NAME) that you just identified. This creates a line that connects the source and target nodes. The copy rule is displayed in the From and To sections at the bottom of the Edit Assign dialog box.

6. Enter third pair of parameters by clicking the Expression icon to invoke the Expression Builder dialog.
 - Enter 'STANDARD' in the Expression box. Click **OK**. The Expression icon with the expression value ('STANDARD') appears in the center of the Edit Assign dialog, between the From and To navigation tree nodes.

- In the To navigation tree, navigate to **Variables > Process > Variables > InvokePurchaseOrderService_PurchaseOrderService_ApprovePurchaseOrder_InputVariable > header > ns5:ServiceBean_Header** and select **ns5:SECURITY_GROUP_NAME**.

Drag the Expression icon to connect to the target node (ns5:SECURITY_GROUP_NAME) that you just identified. This creates a line that connects the source and target nodes. The copy rule is displayed in the From and To sections at the bottom of the Edit Assign dialog box.

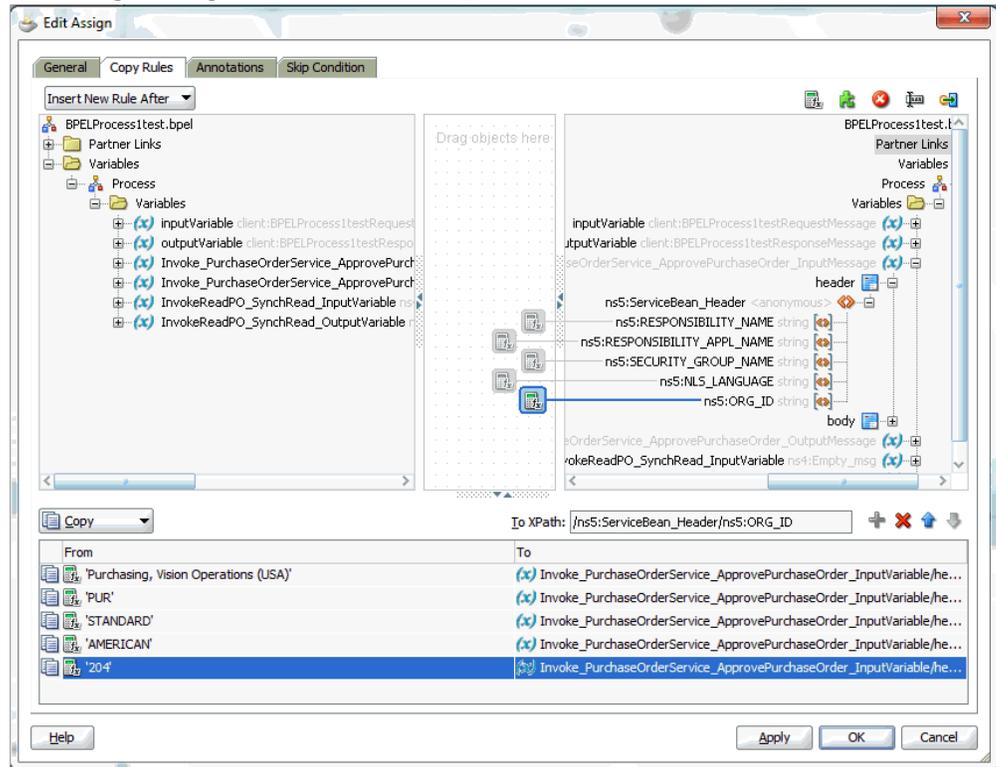
7. Enter the fourth pair of parameters by clicking the Expression icon to invoke the Expression Builder dialog.
 - Enter 'AMERICAN' in the Expression box. Click **OK**. The Expression icon with the expression value ('AMERICAN') appears in the center of the Edit Assign dialog, between the From and To navigation tree nodes.
 - In the To navigation tree, navigate to **Variables > Process > Variables > InvokePurchaseOrderService_PurchaseOrderService_ApprovePurchaseOrder_InputVariable > header > ns5:ServiceBean_Header** and select **ns5:NLS_LANGUAGE**.

Drag the Expression icon to connect to the target node (ns5:NLS_LANGUAGE) that you just identified. This creates a line that connects the source and target nodes. The copy rule is displayed in the From and To sections at the bottom of the Edit Assign dialog box.

8. Enter the fifth pair of parameters by clicking the Expression icon to invoke the Expression Builder dialog.
 - Enter '204' in the Expression box. Click **OK**. The Expression icon with the expression value ('204') appears in the center of the Edit Assign dialog, between the From and To navigation tree nodes.
 - In the To navigation tree, navigate to **Variables > Process > Variables > InvokePurchaseOrderService_PurchaseOrderService_ApprovePurchaseOrder_InputVariable > header > ns5:ServiceBean_Header** and select **ns5:ORG_ID**.

Drag the Expression icon to connect to the target node (ns5:ORG_ID) that you just identified. This creates a line that connects the source and target nodes. The copy rule is displayed in the From and To sections at the bottom of the Edit Assign dialog box.

Edit Assign Dialog with All ServiceBean_Header Parameters



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

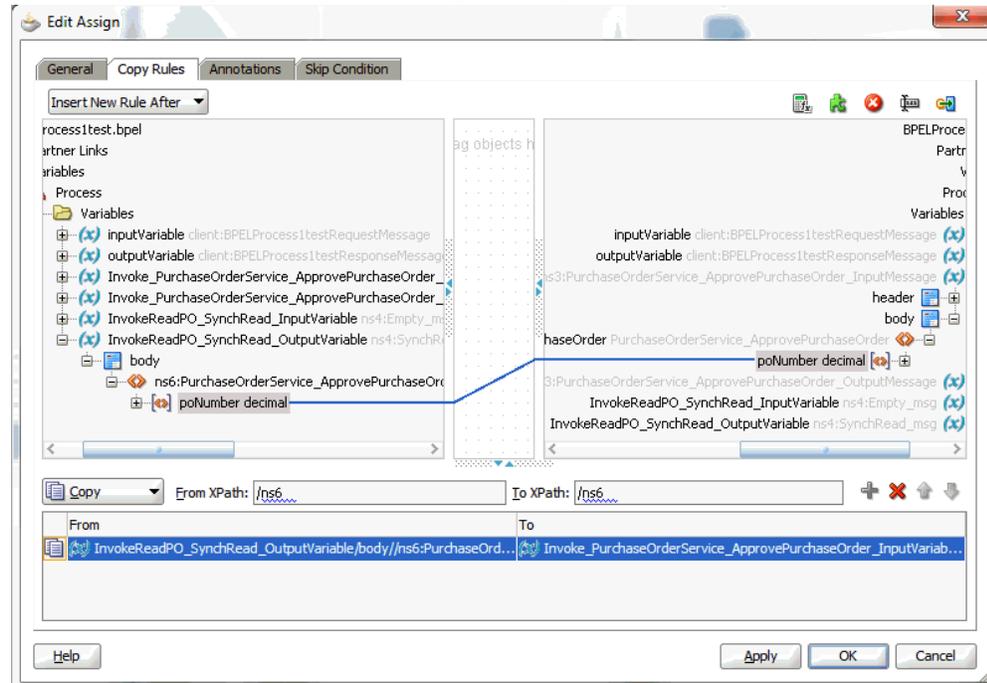
To enter the second Assign activity to pass PO details to the InvokePOService Invoke activity:

1. In Oracle JDeveloper BPEL Designer, drag and drop the second **Assign** activity from the **BPEL Constructs** into the center swim lane of the process diagram, between the first **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. Select the **Copy Rules** tab and expand the target trees:
 - In the **From** navigation tree, navigate to **Variables > Process > Variables > InvokeReadPO_SynchRead_OutputVariable > PurchaseOrderService_ApprovePurchaseOrder > body > ns6:PurchaseOrderService_ApprovePurchaseOrder** and select **poNumber**.
 - In the **To** navigation tree, navigate to **Variables > Process > Variables >**

InvokePurchaseOrderService_PurchaseOrderService_ApprovePurchaseOrder_InputVariable > body > ns6:PurchaseOrderService_ApprovePurchaseOrder and select **poNumber**.

Drag the source node (poNumber) to connect to the target node (poNumber) that you just identified. This creates a line that connects the source and target nodes. The copy rule is displayed in the From and To sections at the bottom of the Edit Assign dialog box.

Edit Assign Dialog for the Second Assign Activity



5. 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:

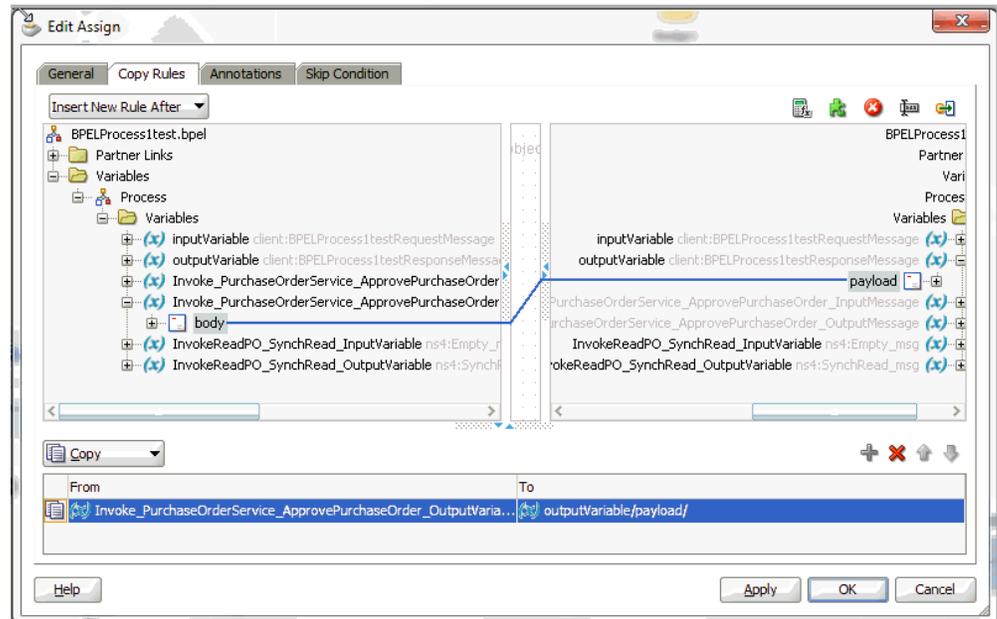
1. Add the third Assign activity by dragging and dropping the **Assign** activity into the center swim lane of the process diagram, between the **InvokePOService Invoke** and the **Replyoutput** activities.
2. Repeat Step 2 to Step 3 described in creating the first Assign activity to add the second Assign activity called 'SetPOStatus'.
3. Select the Copy Rules tab and expand the source and target trees:
 - In the From navigation tree, navigate to **Variables > Process > Variables > InvokePurchaseOrderService_PurchaseOrderService_ApprovePurchaseOrder**

_OutputVariable and select **body**.

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

Drag the source node (body) to connect to the target node (payload) that you just identified. This creates a line that connects the source and target nodes. The copy rule is displayed in the From and To sections at the bottom of the Edit Assign dialog box.

Edit Assign Dialog for the Third Assign Activity



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

Configuring Web Service Policies

Use the following steps to add a security policy at design time:

1. Navigate to SOA Content > Business Rules > composite.xml. Right click on the PurchaseOrderService service and select "Configure WS Policies" from the drop-down list.
2. The Configure SOA WS Policies dialog appears.

In the Security section, click the **Add** icon (+). The Select Server Security Policies dialog appears.

Select 'oracle/wss_username_token_service_policy' and click **OK**.

The attached security policy is shown in the Security section.

3. From the navigation menu, select **View > Property Inspector** to display the Property Inspector window for `PurchaseOrderService` service component.
In the Properties section, click the **Add** icon (+) for binding properties. The Create Property dialog appears.
Enter `'oracle.webservices.auth.username'` in the Name field and enter `'operations'` as the value.
Click **OK**.
4. Use the same approach by clicking the **Add** icon (+) again in the Properties section for binding properties. Enter `'oracle.webservices.auth.password'` in the Name field. Enter the associated password for user `'operations'` in the Value field.
Click **OK**.
Both selected property names and values appear in the Properties section.

Deploying and Testing the SOA Composite with BPEL Process at Runtime

To invoke the synchronous `PurchaseOrderService` from the BPEL client contained in the SOA composite, the SOA composite needs to be deployed to the Oracle WebLogic managed server. This can be achieved using Oracle JDeveloper. Once the composite is deployed, it can be tested from the Oracle Enterprise Manager Fusion Middleware Control Console.

Prerequisites

Before deploying the SOA composite with BPEL process using Oracle JDeveloper, you must have established the connectivity between the design-time environment and the runtime server. For information on how to configure the necessary server connection, see *Configuring Server Connection*, page B-1.

Note: If a local instance of the Oracle WebLogic Server is used, start the WebLogic Server by selecting `Run > Start Server Instance` from Oracle JDeveloper. Once the WebLogic Admin Server "DefaultServer" instance is successfully started, the `<Server started in Running mode>` and `DefaultServer started` message in the `Running:DefaultServer` and `Messages` logs should appear.

Perform the following runtime tasks:

1. Deploy the SOA Composite Application with BPEL Process, page 9-22
2. Test the SOA Composite Application with BPEL Process, page 9-24

Deploying the SOA Composite with BPEL Process

You must deploy the Approve Purchase Order SOA composite (`POApprove.bpel`)

that you created earlier before you can run it.

To deploy the SOA composite application:

1. In the Applications Navigator of JDeveloper, select the **POApprove** project.
2. Right-click the project and select **Deploy > [project name] > [serverConnection]** from the menu.

For example, you can select **Deploy > POApprove > SOAServer** to deploy the process if you have the connection appropriately.

Note: If this is the first time to set up the server connection, then the Deployment Action dialog appears. Select 'Deploy to Application Server' and click **Next**.

In the Deploy Configuration dialog, ensure the following information is selected before clicking **Next** to add a new application server:

- New Revision ID: 1.0
- Mark composite revision as default: Select this checkbox.
- Overwrite any existing composites with the same revision ID: Select this checkbox.

The steps to create a new Oracle WebLogic Server connection from Oracle JDeveloper are covered in Configuring Server Connection, page B-1.

3. In the Select Server dialog, select 'soa-server1' that you have established the server connection earlier. Click **Next**.
4. In the SOA Servers dialog, accept the default target SOA Server ('soa-server1') selection.

Click **Next** and **Finish**.

5. If you are deploying the composite for the first time from your Oracle JDeveloper session, the Authorization Request window appears. Enter the user name and corresponding password specified during the Oracle SOA Suite installation. Click **OK**.
6. Deployment processing starts. Monitor deployment process and check for successful compilation in the SOA - Log window.

Verify that the deployment is successful in the Deployment - Log window.

Testing the SOA Composite Application with BPEL Process

Once the BPEL process contained in the SOA composite has been deployed, you can manage and monitor the process from Oracle Enterprise Manager Fusion Middleware Control Console.

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

To test the SOA composite application with BPEL process:

1. Navigate to Oracle Enterprise Manager Fusion Middleware Control Console (`http://<hostname>:<port>/em`). The login page appears.
2. Enter the user name and corresponding password specified during the installation, and then click **Login** to log in to a farm. The composite (ApprovePO) you deployed is displayed in the Applications Navigation tree.

You may need to select an appropriate target instance farm if there are multiple target Oracle Enterprise Manager Fusion Middleware Control Console farms.

3. From the Farm navigation pane, expand the SOA >soa-infra node in the tree to navigate through the SOA Infrastructure home page and menu to access your deployed SOA composite applications running on soa-infra managed server.

Click the ApprovePO [1.0] link.

4. Click the Policies tab and notice that the 'oracle/wss_username_token_service_policy' policy you attached to the PurchaseOrderService service binding earlier at the design time is now displayed here.
5. In the ApprovePO [1.0] home page, click **Test**.
6. The Test Web Service page for initiating an instance appears. You can specify information as XML payload data to use in the Input Arguments section.

Note: If the WS-Security credentials are not entered at design time, you can enter the credentials at runtime by selecting the WSS Username Token option in the Security section at the top of the Request tab. Enter 'operations' in the Username field and the associated password for user 'operations'.

Click **Test Web Service** to initiate the process.

The test results appear in the Response tab upon completion.

7. Click your BPEL service component instance link (such as ApprovePO) to display the Instances page where you can view the invocation details of the BPEL activities

in the Audit Trail tab.

Click the Flow tab to check the BPEL process flow diagram. Click an activity of the process diagram to view the activity details and flow of the payload through the process.

8. 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 as a user who has the Purchasing responsibility. Open the Purchase Orders form and search for the supplier to bring up the purchase order details. You will 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 9-25
2. Creating a Security Grant for the Deployed Service, page 9-27
3. Recording Resource Information from Deployed WADL, page 9-28
4. Invoking a REST Service Using Command Lines, page 9-28

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 checkboxes.

- In the REST Service Security region, ensure that at least one authentication type is selected.

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

The screenshot shows the 'Interface Details Page' for 'Location Business Object Services' in the Oracle Integration Repository. The page is divided into several sections:

- Header:** 'Integration Repository' and 'Administration' tabs. 'Integration Repository >' breadcrumb.
- Service Information:** 'Business Service Object : Location Business Object Services'. Fields include: Qualified Name (/oracle/apps/ar/hz/service/party/LocationService), Interface (oracle.apps.ar.hz.service.party.LocationService), Extends (oracle.svc.Service), Product (Receivables), and Documentation (Location Business Object API, Oracle Trading Community Architecture Technical Implementation Guide). Status is Active, Scope is Public, and Interface Source is Oracle.
- Navigation:** Overview, SOAP Web Service, REST Web Service (selected), Grants.
- Personalize Stack Layout:** Service Alias (location), REST Service Status (Deployed), View WADL, Design Time Log (disabled), Configure.
- Service Operations:** A table listing operations with columns: Name, Internal Name, GET, POST, Grant.

Name	Internal Name	GET	POST	Grant
Location Business Object Services				
create Location	createLocation			<input type="checkbox"/>
get Location	getLocation	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
save Location	saveLocation		<input checked="" type="checkbox"/>	<input type="checkbox"/>
update Location	updateLocation		<input checked="" type="checkbox"/>	<input type="checkbox"/>

TIP To apply any changes in Operation, Undeploy the service.

REST Service Security

*Authentication Type: HTTP Basic, Security Token

Tip: Use [Login Service](#) to obtain Security Token for given user credentials.

Buttons: Undeploy, Browse, Search, Printable Page.

Copyright (c) 1998, 2020, Oracle and/or its affiliates. All rights reserved. About this Page Privacy Statement

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 checkbox 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.

Fetching Data Using the HTTP GET Method

When a request of fetching location data is received, the GET method of the `getLocation` service operation is used to fetch TCA location details for a particular location. The output response with the location details will be sent back to the requestor.

Invoking the REST Service Using Command Line Tool

For example, run the following `curl` command to fetch `locationId` values through the `getLocation` REST service operation.

```
curl -u <username>:<password> -X GET  
'https://<ebshost>:<port>/webservices/rest/<alias>/getLocation/?  
locationID=<location id>'
```

- Replace `<username>` with an Oracle E-Business Suite user name who has the

privilege of accessing the Location Business Object Services interface.

- Replace <alias> with location in this example.
- Replace <location id> with the actual ID for the invocation.

Viewing the Response Message

When the REST service is successfully invoked, it may return the following response:

```

{
  "tns0:getLocation_Output" : {
    "@xmlns:tns0" : "ht/rest/location/getLocation/",
    "@xmlns:xsi" : "http://www.w3.org/2001/XMLSchema-instance",
    "@xmlns:xsd" : "http://www.w3.org/2001/XMLSchema",
    "@xmlns:tns1" : "http://xmlns.oracle.com/apps/ar/hz/service/party",
    "tns1:OutputParameters" : {
      "location" : {
        "ActionType" : null,
        "LocationId" : "xxx",
        "OrigSystem" : null,
        "OrigSystemReference" : null,
        "Country" : "US",
        "Address1" : "401 Island Parkway",
        "Address2" : null,
        "Address3" : "Example Inc.",
        "Address4" : null,
        "City" : "Redwood Shores",
        "PostalCode" : "94065",
        "State" : "CA",
        "Province" : null,
        "County" : "San Mateo",
        "AddressKey" : null,
        "AddressStyle" : null,
        "ValidatedFlag" : null,
        "AddressLinesPhonetic" : null,
        "PostalPlus4Code" : null,
        "Position" : null,
        "LocationDirections" : null,
        "AddressEffectiveDate" : null,
        "AddressExpirationDate" : null,
        "ClliCode" : null,
        "Language" : null,
        "ShortDescription" : null,
        "Description" : null,
        "LocHierarchyId" : null,
        "SalesTaxGeocode" : null,
        "SalesTaxInsideCityLimits" : null,
        "FaLocationId" : null,
        "TimezoneId" : "1",
        "ProgramUpdateDate" : "2005-05-04T17:49:58.0",
        "CreatedByModule" : null,
        "CreatedByName" : "xxxxxx",
        "CreationDate" : "1997-02-21T00:00:00.0",
        "LastUpdateDate" : "2005-05-04T17:49:58.0",
        "LastUpdatedByName" : "xxxxxx",
        "ActualContentSource" : "USER_ENTERED",
        "DeliveryPointCode" : null,
        "GeometryStatusCode" : "MULTIMATCH",
        "CommonObjId" : null,
        "Geometry" : null
      }
    }
  }
}

```

Using Composite Services - BPEL

Overview

A composite service is a coarse-grained abstracted service. It uses native services as building blocks to orchestrate the business invocation sequence into a meaningful end-to-end business flow through a web service composition language BPEL.

At design time, integration developers use BPEL process component in Oracle JDeveloper 10g to assemble a series of service components together for a business function, and then they annotate the composite service - BPEL definition based on the Integration Repository annotation standards. After validation, the newly-created composite service - BPEL can be uploaded to Integration Repository where it can be displayed and available to all users.

The integration developer can view each composite service - BPEL details by selecting a desired service from the Oracle Integration Repository browser, downloading the selected composite service - BPEL to a local directory, modifying the BPEL process in Oracle JDeveloper 10g if it's necessary, and deploying it to a BPEL server in Oracle SOA Suite 10g BPEL Process Manager or a third party BPEL PM in a J2EE environment.

This chapter explains how to view, download, and modify composite services - BPEL. Detailed design-time tasks on how to create a SOA composite service with BPEL process are included in each individual interface described earlier in this book.

- Viewing Composite Services - BPEL, page 10-2
- Downloading Composite Services - BPEL, page 10-2
- Modifying and Deploying BPEL processes, page 10-3

For information on how to annotate composite - BPEL definitions, see Composite Service - BPEL Annotations, page A-115.

For information on how to upload composite - BPEL definitions to the Integration Repository, see Enabling Custom Integration Interface Process Flow, *Oracle E-Business*

Viewing Composite Services - BPEL

Once annotated custom composite - BPEL definitions are uploaded to the Integration Repository, 'Composite - BPEL' option can be displayed from the repository and available to all users.

An integration developer can view a composite service - BPEL by navigating to the Composite - BPEL interface type directly from the Oracle Integration Repository Browser window with View By 'Interface Type'. Alternatively, the integration developer can perform a search to locate the composite service - BPEL in the Search page.

To view a composite service - BPEL details, click on a composite service name link from the navigation tree or search results. The composite service - BPEL interface details page is displayed with service name, description, BPEL file, WSDL file, and other annotated information.

The composite service - BPEL details page allows you to perform the following tasks in the BPEL Files region:

- View an abstract WSDL file by clicking the **View Abstract WSDL** link
See: Reviewing Web Service WSDL Source, *Oracle E-Business Suite Integrated SOA Gateway User's Guide*.
- View the composite - BPEL file by clicking the **View BPEL File** link
The BPEL code is displayed in a pop-up window containing major BPEL process components and activities included for the selected composite service.

For information on how to annotate composite - BPEL definitions, see Composite Service - BPEL Annotations, page A-115.

For information on how to upload composite - BPEL definitions to the Integration Repository, see Enabling Custom Integration Interface Process Flow, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

You can download a corresponding composite service BPEL project file to your local machine. See: Downloading Composite Services - BPEL, page 10-2.

Downloading Composite Services - BPEL

In addition to viewing composite service - BPEL details and reviewing a WSDL abstract, an integration developer can download the composite service relevant files aggregated in a .JAR file to a local directory.

Important: In general, only integration developers and integration

administrators can download the composite services - BPEL. However, users who are granted the download privilege through Integration Repository Download Composite Service permission set (FND_REP_DOWNLOAD_PERM_SET) can also perform the download action. Otherwise, **Download Service** may not appear in the details page by default.

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*.

To download the .ZIP file for a composite service - BPEL, navigate to the composite service - BPEL details page for a service that you want to download, and then click **Download Service** to download the file to your local directory.

After the download, you can unzip the BPEL .JAR file and open the BPEL process in Oracle JDeveloper 10g for modification on service endpoints if needed. For information on how to modify and deploy BPEL process, see Modifying and Deploying BPEL processes, page 10-3.

To download a composite service - BPEL:

1. Log in to Oracle E-Business Suite as a user who has the Integration Developer role. Select the Integrated SOA Gateway responsibility and choose the Integration Repository link from the navigation menu.
2. In the Integration Repository tab, select 'Interface Type' from the View By drop-down list.
3. Expand the Composite - BPEL interface type node to locate your desired composite service - BPEL.
4. Click the composite service - BPEL that you want to download to open the Composite Service - BPEL interface details page.
5. Click **Download Service** to download the selected composite BPEL file to your local directory.

Modifying and Deploying BPEL Processes

After downloading a composite service BPEL project, an integration developer can modify the BPEL project if it's needed. This can be done by first unzipping the BPEL .JAR file and then opening the BPEL file in Oracle JDeveloper 10g to modify the BPEL process endpoints if it's needed. After the modification, the BPEL project can be deployed to a BPEL server in Oracle SOA Suite 10g.

Note: In Oracle SOA Suite 11g and Oracle SOA Suite 12c, BPEL process is managed and deployed together with the associated SOA composite application.

Composite service - BPEL type is supported in Oracle SOA Suite 10g. The BPEL process is deployed to a BPEL server in Oracle SOA Suite 10g or a third party BPEL PM in a J2EE environment.

The modification of a BPEL process uses the similar logic during the BPEL process creation. Refer to the design-time tasks for each interface type discussed earlier in this book. For information on how to test and validate the BPEL process within a composite application, refer to the runtime tasks of the interface type described in this book.

For BPEL process modification and deployment described in this section, Oracle JDeveloper 10g (10.1.3.3.0) is used as a design-time tool to modify the BPEL process, and Oracle SOA Suite BPEL server 10.1.3.3.0 is used for the process deployment.

To modify a BPEL process:

1. Open a BPEL file in Oracle JDeveloper 10g 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 Oracle JDeveloper 10g 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 project if you have the BPEL Process Manager set up 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.

Creating and Using Custom Integration Interfaces

Overview

Custom integration interfaces and services can be viewed and displayed through the Integration Repository browser window along with Oracle seeded interfaces in Oracle E-Business Suite.

Integration developers create and annotate custom integration interfaces based on the Integration Repository annotation standards for the supported interface types including XML Gateway Map, Business Event, PL/SQL, Concurrent Program, Business Service Object, Java Bean Services, Application Module Services, Java APIs, and Composite Service - BPEL type.

Note: Please note that custom interface types of EDI, Open Interface Tables, Open Interface Views, and Java APIs for Forms interfaces are not supported in this release.

Oracle Integration Repository currently does not support the creation of custom Product Family and custom Business Entity.

Once these custom interfaces are annotated, integration administrators use a standalone design-time tool to validate these annotated source files against the annotation standards. After validation, a loader file is generated and then uploaded to the Integration Repository through backend processing. These custom interfaces are displayed based on the interface types to which they belong and displayed together with Oracle seeded ones from the Integration Repository user interface.

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 administrator should first create necessary security grants, and then generate and deploy the web service 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 services - BPEL type

If a custom interface is needed for a composite service - BPEL type, an integration developer will first create a composite service by orchestrating discrete native services into a meaningful process flow using BPEL process component in Oracle JDeveloper 10g. Based on the annotation standards specifically for composite service, the integration developer will then annotate the composite service - BPEL type. Similar to custom interfaces of other interface types, appropriate validation on the BPEL project JAR file is required before it can be 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, page 11-2
- Using Custom Integration Interfaces as Web Services, page 11-24

Creating Custom Integration Interfaces

The following topics are discussed in this section:

- Creating Custom Integration Interfaces of Native Interface Types, page 11-2
- Creating Custom Composite Services, page 11-10
- Creating Custom Business Events Using Workflow XML Loader, page 11-16

Creating Custom Integration Interfaces of Native Interface Types

Custom interface definitions can be created and annotated for almost all interface types. With appropriate validation, if no error occurred, the validated custom definition sources compiled in a generated iLDT file can be uploaded to Oracle Integration Repository through backend processing.

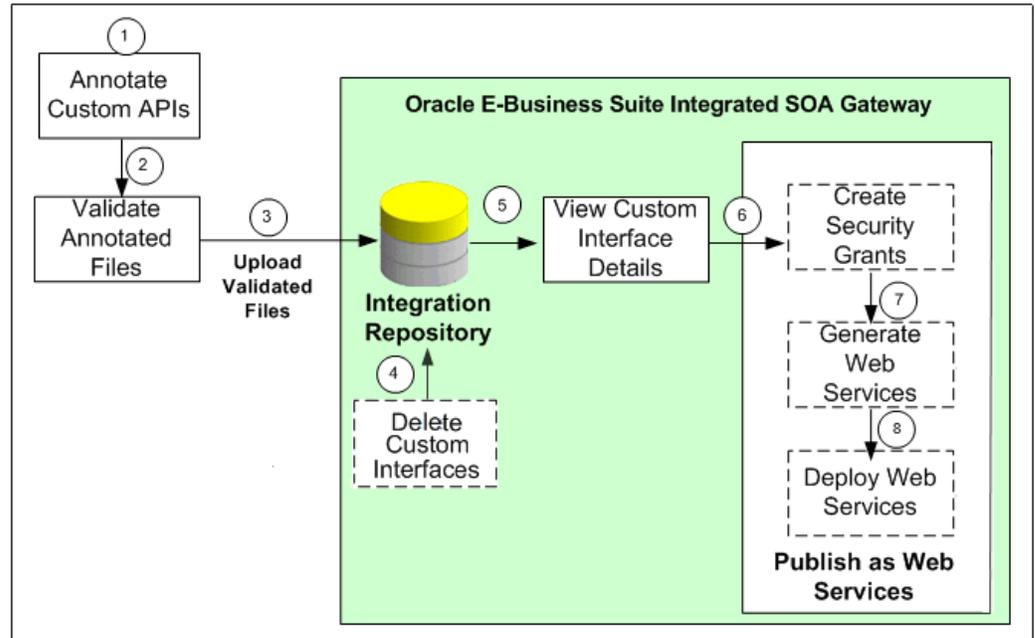
Note: Custom interface types of EDI, Open Interface Tables, and Open Interface Views are not supported in this release.

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:

Custom Integration Interfaces Development Process Flow



1. Users who have the Integration Developer role 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.

Note: For custom PL/SQL APIs (simple data types only) that are created with a custom schema, you can publish such custom APIs in Oracle Integration Repository. Additionally, perform the following tasks for such APIs with a custom schema:

1. Grant access to APPS schema.
 1. Connect to a custom schema as EBS_SYSTEM if your instance is on AD and TXK Delta 13 release update packs (RUPs) or later, or as SYSTEM if your instance is on an earlier AD and TXK RUP:

```
sqlplus '/ as EBS_SYSTEM'
```

Note: The R12.AD.C.Delta.13 and R12.TXK.C.Delta.13 RUPs introduce the EBS_SYSTEM schema. If you are running Release Update Packs for AD and TXK

Delta 13 or later, database privileges are granted to the Oracle E-Business Suite administration account, EBS_SYSTEM. Only the minimally required database privileges required to run Oracle E-Business Suite are granted to APPS by EBS_SYSTEM. For more information, refer to:

- Document 2755875.1, *Oracle E-Business Suite Release 12.2 System Schema Migration*
- Document 2758993.1, *Managing Database Privileges in Oracle E-Business Suite Release 12.2 (Running adgrants.sql)*

2. Use the following command to grant access:

```
GRANT EXECUTE on <custom_schema> .  
<custom_package> TO APPS;
```

2. Create a synonym for the custom stored procedure.

1. Connect to APPS schema using the following command:

```
sqlplus <APPS Username>  
Enter password: password
```

2. Use the following command to create a synonym:

```
CREATE SYNONYM <custom_package> FOR  
<custom_schema> .<custom_package>;
```

2. Users who have the Integration Administrator role validate the annotated custom interface definitions against the annotation standards. This validation is performed by running 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 Administrator role 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 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 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 generated or deployed, it must be reset or undeployed to its initial state before it can be deleted. 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 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 Administrator role can generate web 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** 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 Administrator role deploy the web services from Oracle Integration Repository to the application server.

To deploy the generated SOAP web services, the administrators must first select one authentication type (Username Token or SAML Token) for each selected web service and then click **Deploy** in the selected interface details page. This deploys the generated service with 'Active' state to Oracle SOA Suite where Oracle E-Business Suite services can be exposed as standard web services for service invocation at runtime. 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 and specify the desired service operations before deploying the service. Additionally, the administrators need to specify HTTP methods for the service operations contained in the selected interface if it is an interface type of PL/SQL, Java Bean Services, Application Module Services, or Business Service Object.

Note: Although open interface tables and open interface views can be exposed as REST services, custom open interface tables and custom open interface views 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

Once a custom integration 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 custom interface 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. For information on how to annotate custom interfaces for the interface types supported by Oracle Integration Repository, see *Integration Repository Annotation Standards*, page A-1 and *Oracle Application Framework Developer's Guide*, available from My Oracle Support Knowledge Document 1315485.1.

```

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 running 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, and IBM AIX on Power Systems.

Microsoft Windows platform is currently not supported in this release.

Once an iLDT file is generated, an integration administrator can upload the generated file to Oracle Integration Repository where the custom interfaces can be exposed to all users.

For information on how to set up and use the Integration Repository Parser to generate the iLDT file as well as how to upload the generated file to the repository, 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 Custom Interfaces and Performing Administrative Tasks

Searching and Viewing Custom Interfaces

Once annotated custom interface definitions have been uploaded successfully, they are merged into the interface types they belong to and displayed together with Oracle seeded interfaces from the Integration Repository browser window. To easily distinguish annotated custom interface definitions from Oracle ones, the Interface Source "Custom" is used to categorize those custom interfaces in contrast to Interface Source "Oracle" for Oracle seeded interfaces in Oracle E-Business Suite.

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 you perform 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 have been successfully uploaded and displayed from the Integration Repository user interface, an integration administrator can perform the same administrative tasks on these custom interfaces as they are for the native integration interfaces. These administrative tasks including creating security grants for newly created custom interfaces if needed, generating web services, deploying web services, and managing services throughout the entire deployment life cycle. See *Administering Custom Integration Interfaces and Services, Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

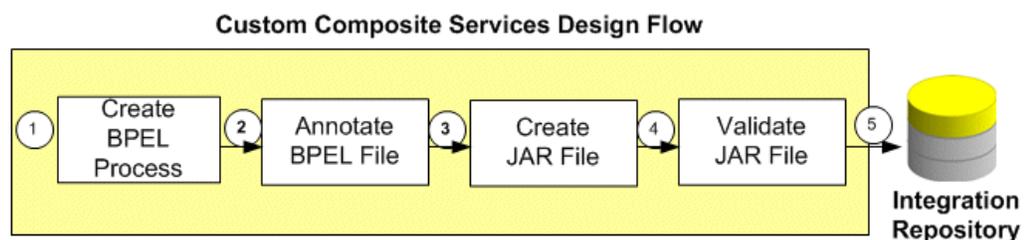
For information on how to use custom integration interfaces as web services, see *Using Custom Integration Interfaces as Web Services, page 11-24*.

Creating Custom Composite Services - BPEL

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 service - BPEL type and validation against the annotation standards, if no error occurred, an iLDT file can be generated for the validated composite service BPEL project. The generated iLDT file can then be uploaded to the Integration Repository through backend processing.

Creating Custom Composite Services

The following diagram illustrates the custom integration interface design flow for composite service - BPEL type:



1. An integration developer orchestrates a composite service using BPEL process component in Oracle JDeveloper 10g.

2. An integration developer annotates the composite service based on the Integration Repository annotation standards specifically for the composite service - BPEL type.
See: *Creating and Annotating Custom Composite Services - BPEL*, page 11-11.
3. An integration developer creates a JAR file of the composite service - BPEL project.
4. An integration administrator unzips the JAR file first and then validates the annotated custom interface definitions against the annotation standards specifically for composite services - BPEL. This validation is performed by running the Integration Repository Parser to read the annotated files and then generate an Integration Repository loader file (iLDT) if no error occurred.
5. An integration 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, verify the custom composite service - BPEL from the Integration Repository user interface.

Once custom integration interface definitions have been successfully uploaded and displayed from the Integration Repository browser, the integration administrator and the integration developer can download the composite services - BPEL for modification if needed. For information on how to download composite services - BPEL, see *Viewing and Downloading Custom Composite Services - BPEL*, page 11-15.

A Custom Composite Service Annotation Example

The key essence of creating custom integration interfaces relies on properly explanation of the new interface feature or definition. When a custom composite service - BPEL definition is created, an integration developer must properly annotate the custom source file based on the Integration Repository annotation standards so that the source file can be displayed with appropriate description from the browser interface.

For example, a Create Invoice composite - BPEL project is created. To annotate the composite service, open the *.bpe1 file in text editor and place the annotation within the comments section in the beginning of the file as highlighted here:

```

////////////////////////////////////
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/soapprovider/plsql/AR_INVOICE_API_PUB_2108/CREATE_SINGLE_INVOICE_1037
895/"
  xmlns:ns3="http://xmlns.oracle.
com/soapprovider/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"/>

```

```

<partnerLink 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-115.

Generating and Uploading iLDT Files

Once annotated custom composite services - BPEL have been 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 running 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, and IBM AIX on Power Systems.

Microsoft Windows platform is currently not supported in this release.

Once an iLDT file has been successfully generated, an integration administrator can upload the generated file to Oracle Integration Repository where the custom interfaces can be exposed to all users.

For information on how to set up and use the Integration Repository Parser to generate an iLDT file as well as how to upload the generated 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 - BPEL

Once annotated custom composite service definitions have been successfully uploaded to the Integration Repository user interface, they are merged into the Composite Service BPEL type and displayed together with Oracle seeded interfaces. To easily distinguish custom composite services from Oracle seeded ones, Interface Source "Custom" is used to categorize those custom interfaces in contrast to Interface Source "Oracle" for Oracle seeded interfaces in Oracle E-Business Suite.

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 you perform the search, all matched custom composite services will be displayed.

Downloading Custom Composite Services - BPEL

Similar to downloading native packaged composite services, integration administrators and integration developers can click **Download Service** in the composite service - BPEL interface details page to download the relevant custom composite files aggregated in a .JAR file to their local directories.

For more information on how to search and download custom composite services -

BPEL, see Downloading Composite Services - BPEL, *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, download the events that you have created, annotate the event source codes, validate the source 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, you first locate them and then download them using Workflow XML Loader.

Events Page with Search and Results Regions

The screenshot shows the 'Business Events' page in Oracle Developer Studio. The page has a navigation bar with 'Home', 'Developer Studio', 'Business Events', 'Status Monitor', 'Notifications', and 'Administration'. Below the navigation bar, there are tabs for 'Events', 'Subscriptions', 'Agents', and 'Systems'. The main content area is titled 'Business Events: Events >' and 'Events'. It includes a search section with a text input field containing '%oracle.apps.ecx.inbound%' and a 'Go' button. Below the search section is a 'Results: Events' section with a table of search results. The table has columns for 'Name', 'Display Name', 'Type', 'Status', 'Subscription', 'Update', and 'Test'. There are three rows of results, each with a checkbox in the 'Select' column and icons for 'Subscription', 'Update', and 'Test'.

Select	Name ▲	Display Name ▲	Type ▲	Status ▲	Subscription	Update	Test
<input type="checkbox"/>	oracle.apps.ecx.inbound.message.process	Generic Inbound Message Process Event	Event	Enabled			
<input type="checkbox"/>	oracle.apps.ecx.inbound.message.receive	Generic Inbound Message Event	Event	Enabled			
<input type="checkbox"/>	oracle.apps.ecx.inbound.process_at_java	Oracle XML Gateway processing inbound messages at middle tier event	Event	Enabled			

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 `wfdemo.e.wfx`. In the filename, the first two or three characters refer 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 wfdemo.e.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 wfdemo.e.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 `wfdemo.e.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 in to Oracle E-Business Suite as a user who has 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-132, 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 run 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, run 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 current
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 running the parser, you should 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 occurred in the annotated interface file, an iLDT (*.ildt) file would be generated. An integration 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 running the Integration Repository Parser, you need to install `perl` modules and apply necessary patches. For the 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 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:12345:sid100 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:12345:sid100 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 occurred during the iLDT generation. In order for users to view the custom business events through the Integration Repository, an integration administrator needs to manually upload the generated iLDT file to the Integration Repository using FNDLOAD command.

```
$FND_TOP/bin/FNDLOAD <APPS username> 0 Y UPLOAD  
$fnd/patch/115/import/wfirep.lct <ildt file>  
ORACLE Password:
```

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 have been successfully uploaded, are merged into the interface types they belong to and displayed together with Oracle seeded interfaces from the Integration Repository browser window. To easily distinguish 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 seeded interfaces in Oracle E-Business Suite.

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 you perform 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 have been successfully uploaded and displayed from the Integration Repository browser window, an integration administrator can perform the same administrative tasks on these custom events as they are for the native events. 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*.

Using Custom Integration Interfaces as Web Services

Overview

With appropriate annotation and validation, a custom integration interface can be created 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 has been 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 seeded 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 runtime tasks in creating and

deploying a SOA composite application with BPEL process are discussed in this section. For the example described in the following sections, Oracle JDeveloper 11g (11.1.1.6.0) is used as a design-time tool to create the SOA composite application with BPEL process and Oracle SOA Suite 11g (11.1.1.6.0) is used for the process deployment.

Note: While using Oracle JDeveloper with other Oracle Fusion Middleware components (such as Oracle SOA Suite), to enable SOA technologies, you need to manually download Oracle SOA Suite Composite Editor, an Oracle JDeveloper extension for SOA technologies. For more information on installing additional Oracle Fusion Middleware design-time components, see the *Oracle Fusion Middleware Installation Guide for Oracle JDeveloper*.

Using Custom Interface WSDL in Creating a SOA Composite Application with BPEL Process at Design Time

SOA Composite Application with BPEL Process Scenario

This example uses a custom PL/SQL API `ZZ_SDREQUEST` to explain the BPEL process creation in a SOA Composite application.

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. After the supplier ship and debit request for a product is created, the request number will then be returned to the requestor.

When the SOA composite application with BPEL process has been successfully invoked after deployment, 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 a custom interface exposed as a web service, you need to ensure the following tasks are in place:

- An integration administrator or an integration developer needs to generate a web service first. The administrator will deploy the generated custom service to an Oracle SOA Suite WebLogic managed 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 for a 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 on the custom interface so that users with appropriate privileges can have access to the interface.

For example, the administrator can grant the custom API access privilege to a user with

the Oracle Trade Management responsibility. After the invocation of this custom API, the user can log on to Oracle Trade Management and verify the supplier and debit request creation details.

For information on how to create security grants, see *Creating Grants, Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

Deploying a Web Service Composite for a Custom Interface

An integration administrator or an integration developer must first create a web service for a selected custom interface, and then the administrator can deploy the custom service from Oracle Integration Repository to an Oracle SOA Suite WebLogic managed server.

For example, the following steps must be performed first before the integration developer creates a BPEL process by using the deployed WSDL:

1. To generate a web service, the integration administrator or integration developer locates the interface definition first (such as a custom PL/SQL interface ZZ_SDREQUEST) and selects desired interaction pattern information from the Interaction Pattern table. This can be selected at the interface level or at the method level before clicking **Generate** in the interface details page.

Once the service has been successfully generated, the Web Service Status field changed from 'Not Generated' to 'Generated' in the interface details page. For detailed instructions 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, the integration administrator selects one authentication type before clicking **Deploy**. The deployed service in Oracle SOA Suite is an active service and is ready to accept new SOAP requests.

Once the service has been successfully deployed, the selected authentication type will be displayed along with 'Deployed' with 'Active' state in the Web Service Status field. 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 information 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 a WSDL URL

Apart from the required tasks performed by the administrators, the integration developer needs 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.

This can be done by clicking the **View WSDL** link in the interface details page. Copy the WSDL URL from the new pop-up window. This URL will be used later in creating a partner link service in a BPEL process.

For information on how to search for an interface and view the interface details, see *Searching and Viewing Integration Interfaces*, page 2-1.

SOA Composite Application with 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 SOA Composite Application with BPEL Process**, page 11-27

Use this step to create a new SOA composite application with BPEL process 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 to the request application.

2. **Create a Partner Link**, page 11-29

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 a web service.

3. **Add a Partner Link for File Adapter**, page 11-29

Use this step to synchronously read input data details passed from the first Assign activity to create supplier ship and debit request.

4. **Add Invoke Activities**, page 11-33

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

- Point to the File Adapter to synchronously 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 11-34

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 how to create SOA composite applications using BPEL process service component, see the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite* for details.

Creating a New SOA Composite Application with BPEL Process

Use this step to create a new SOA composite application that will contain various BPEL

process activities.

To create a new SOA composite application with BPEL process:

1. Open Oracle JDeveloper.
2. Click **New Application** in the Application Navigator.
The "Create SOA Application - Name your application" page is displayed.
3. Enter an appropriate name for the application in the Application Name field and select **SOA Application** from the Application Template list.
Click **Next**. The "Create SOA Application - Name your project" page is displayed.
4. Enter an appropriate name for the project in the Project Name field, for example `ZZ_CreateSingle_ShipDebitRequest`.
5. In the Project Technologies tab, select 'Web Services' and ensure that **SOA** is selected from the Available technology list to the Selected technology list.
Click **Next**. The "Create SOA Application - Configure SOA settings" page is displayed.
6. Select **Composite With BPEL Process** from the Composite Template list, and then click **Finish**. You have created a new application, and a SOA project. This automatically creates a SOA composite.
The Create BPEL Process page is displayed.
7. Leave the default **BPEL 1.1 Specification** selection unchanged. This creates a BPEL project that supports the BPEL 1.1 specification.
Enter an appropriate name for the BPEL process in the Name field, for example `ZZ_CreateSingle_ShipDebitRequest`.
Select **Synchronous BPEL Process** in the Template field.
Select **required** from the Transaction drop-down list. Click **OK**.
A synchronous BPEL process is created with the Receive and Reply activities. The required source files including `bpel` and `wSDL`, using the name you specified (for example, `ZZ_CreateSingle_ShipDebitRequest.bpel` and `ZZ_CreateSingle_ShipDebitRequest.wSDL`) and `composite.xml` are also generated.

Note: Service 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. Navigate to SOA Content > Business Rules and double click the `composite.xml` to view the composite diagram.

Double click on the `ZZ_CreateSingle_ShipDebitRequest` component to open the BPEL process.

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 Oracle JDeveloper, place your mouse in the Partner Links area and right click to select **Create Partner Link...** from the pull-down menu. Alternatively, you can drag and drop **Partner Link** from the **BPEL Constructs** list into the right Partner Link swim lane of the process diagram.

The Create Partner Link window appears.

2. Copy the WSDL URL corresponding to the custom service, `ZZ_SDREQUEST`, that you recorded earlier from the Integration Repository in the WSDL File field.

Press the **[Tab]** key.

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**. You can manually enter the partner link name.

4. Select the Partner Link Type and Partner Role values from the drop-down lists.

Click **Apply**.

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

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

The Partner Link `ZZ_SDREQUEST` is added to the Partner Links section in the BPEL process diagram.

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 Oracle JDeveloper, drag and drop the **File Adapter** service from the **BPEL Services** list into the right Partner Link swim lane 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`.
4. Click **Next**. The Adapter Interface dialog box appears.
5. Select the **Define from operation and schema (specified later)** radio button and click **Next**. The Operation dialog box appears.
6. 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

Adapter Configuration Wizard - Step 5 of 8

File Directories

Enter directory information for the incoming file of the Synchronous Read File operation.

Directory names are specified as: Physical Path Logical Name

Directory for Incoming Files (physical path):

Archive processed files
 Archive Directory for Processed Files (physical path):

Delete files after successful retrieval

7. Select the **Physical Path** radio button and enter the physical path for incoming file directory information. For example, enter `/usr/tmp/` as the directory name.

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

) to the specified directory.

Alternatively, click **Browse** to locate the incoming file directory information.

Click **Next** to open the File Name dialog box.

8. Enter the name of the file for the synchronous read file operation. For example, enter `Inputzzsdrequest.xml`. Click **Next**. The Messages dialog box appears.
9. Select **Browse for schema file** in front of the URL field.

The Type Chooser window is displayed.

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

Enter the schema location for the service. Such as `http:`

```
//<soa_suite_hostname>:<port>/soa-  
infra/services/default/<jndi_name>_PLSQL_ZZ_SDREQUEST/ZZ_SDREQ  
UEST_Service?  
XSD=xsd/APPS_XX_BPEL_ZZ_CREATE_SDREQUEST_RE_ZZ_SDREQUEST_ZZ_CR  
EATE_SDREQU.xsd.
```

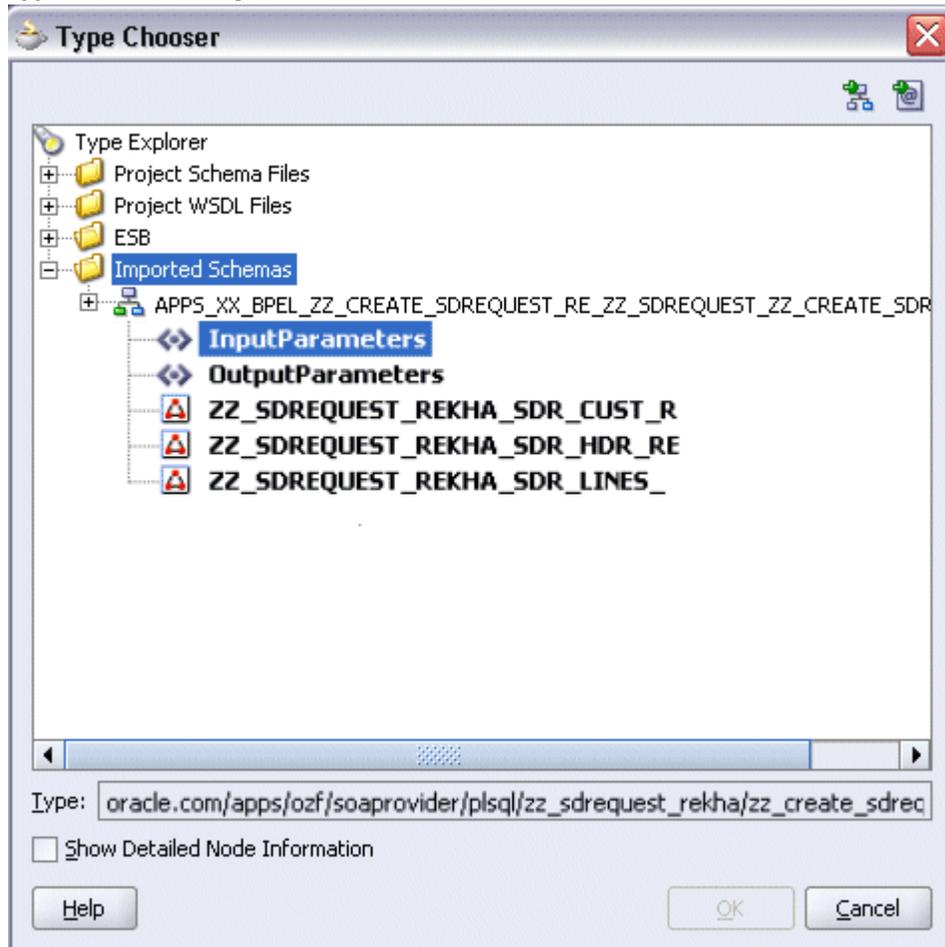
Schema location for your service can be found from the service WSDL URL (for example, `http://<soa_suite_hostname>:<port>/soa-infra/services/default/<jndi_name>_PLSQL_ZZ_SDREQUEST/ZZ_SDREQUEST_Service/?wsdl`).

Select the **Copy to Project** checkbox and click **OK**.

The Localize Files window appears. Ensure the **Maintain original directory structure for imported files** checkbox is selected and click **OK**.

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

Type Chooser Dialog



Expand the Imported Schemas folder and select InputParameters Message in the APPS_XX_BPEL_ZZ_CREATE_SDREQUEST_RE_ZZ_SDREQUEST_ZZ_CREATE_SDR EQU .xsd. Click **OK**.

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

10. 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.

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:

11. Under applications window, navigate to file ReadPayload_file.jca. Set value of property "DeleteFile" to "false".

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 the Read_Payload Partner Link:

1. In Oracle JDeveloper, expand the **BPEL Constructs** from the Component Palette. Drag and drop the **Invoke** activity into the center swim lane of the process diagram, between the **receiveInput** and **replyOutput** 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.
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.

Create Variable Dialog



The screenshot shows a 'Create Variable' dialog box. The 'Name' field contains 'Invoke_Readpayload_InputVariable'. The 'Type' field contains '{http://xmlns.oracle.com/pcbpel/adapter/file/Read_payload/}t'. The 'Global Variable' radio button is selected. The 'OK' and 'Cancel' buttons are at the bottom right.

4. Enter a name for the variable such as 'Invoke_Readpayload_InputVariable' and select the **Global Variable** radio button. Click **OK** in the Create Variable dialog box. Click the **Create** icon next to the **Output Variable** field to create a new variable. The Create Variable dialog box appears.

Enter a name for the output variable such as 'Invoke_Readpayload_OutputVariable' and select the **Global Variable** radio button. 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 the ZZ_SDREQUEST Partner Link:

1. In Oracle JDeveloper, expand the **BPEL Constructs** from the Component Palette. Drag and drop the **Invoke** activity into the center swim lane of the process diagram, after the first **Invoke** activity and the **replayOutput** activity.
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 the **Global Variable** radio button 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 the **Global Variable** radio button 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. Set the application context information obtained from the dummy **Receive** activity. This information 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 application context during service invocation. These SOAHeader elements are *Responsibility*, *RespApplication*, *SecurityGroup*, *NLSLanguage*, and *Org_Id*.

2. Pass the payload information to the `Invoke_zzsdrequest` Invoke activity.
3. Pass the supplier ship and debit request number information to the `Invoke_zzsdrequest` **Invoke** activity.
4. Pass the supplier ship and debit request number information to the dummy **Reply** activity as an output.

To add the first Assign activity to pass application context details to the Invoke_Readpayload Invoke activity:

1. In Oracle JDeveloper, expand the **BPEL Constructs** from the Component Palette. Drag and drop the **Assign** activity into the center swim lane of the process diagram between the **ReceiveInput** activity and the first **Invoke** 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. Select the Copy Rules tab and expand the target trees:
 - Click the Expression icon to invoke the Expression Builder dialog.
Enter '204' in the Expression box. Click **OK**. The Expression icon with the expression value ('204') appears in the center of the Edit Assign dialog, between the From and To navigation tree nodes.
 - In the To navigation tree, navigate to **Variable > Process > Variables > Invoke_zzsrequest_InputVariable >header > ns5:SOAHeader** and select **ns5:ORG_ID**.

Drag the Expression icon to connect to the target node (ns5:ORG_ID) that you just identified. This creates a line that connects the source and target nodes. The copy rule is displayed in the From and To sections at the bottom of the Edit Assign dialog box.

5. Enter the second pair of parameters by clicking the Expression icon to invoke the Expression Builder dialog.
 - Enter 'TRADE_MANAGEMENT_USER' in the Expression box. Click **OK**. The Expression icon with the expression value appears in the center of the Edit Assign dialog, between the From and To navigation tree nodes.
 - In the To navigation tree, navigate to **Variable > Process > Variables > Invoke_zzsrequest_InputVariable >header > ns5:SOAHeader** and select **ns5:Responsibility**.

Drag the Expression icon to connect to the target node (ns5:Responsibility) that you just identified. This creates a line that connects the source and target nodes. The copy rule is displayed in the From and To sections at the bottom of the Edit Assign dialog box.

6. Enter the third pair of parameters by clicking the Expression icon to invoke the Expression Builder dialog.
 - Enter 'OZF' in the Expression box. Click **OK**. The Expression icon with the expression value ('OZF') appears in the center of the Edit Assign dialog,

between the From and To navigation tree nodes.

- In the To navigation tree, navigate to **Variable > Process > Variables > Invoke_zzsdrequest_InputVariable >header > ns5:SOAHeader** and select **ns5:RespApplication**.

Drag the Expression icon to connect to the target node (ns5:RespApplication) that you just identified. This creates a line that connects the source and target nodes. The copy rule is displayed in the From and To sections at the bottom of the Edit Assign dialog box.

7. Enter the fourth pair of parameters by clicking the Expression icon to invoke the Expression Builder dialog.
 - Enter 'STANDARD' in the Expression box. Click **OK**. The Expression icon with the expression value ('STANDARD') appears in the center of the Edit Assign dialog, between the From and To navigation tree nodes.
 - In the To navigation tree, navigate to **Variable > Process > Variables > Invoke_zzsdrequest_InputVariable >header > ns5:SOAHeader** and select **ns5:SecurityGroup**.

Drag the Expression icon to connect to the target node (ns5:SecurityGroup) that you just identified. This creates a line that connects the source and target nodes. The copy rule is displayed in the From and To sections at the bottom of the Edit Assign dialog box.

8. Enter the fifth pair of parameters by clicking the Expression icon to invoke the Expression Builder dialog.
 - Enter 'AMERICAN' in the Expression box. Click **OK**. The Expression icon with the expression value ('AMERICAN') appears in the center of the Edit Assign dialog, between the From and To navigation tree nodes.
 - In the To navigation tree, navigate to **Variable > Process > Variables > Invoke_zzsdrequest_InputVariable >header > ns5:SOAHeader** and select **ns5:NLSLanguage**.

Drag the Expression icon to connect to the target node (ns5:NLSLanguage) that you just identified. This creates a line that connects the source and target nodes. The copy rule is displayed in the From and To sections at the bottom of the Edit Assign dialog box.

9. 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 **BPEL Constructs** into the process diagram, between two **Invoke** activities.

2. Repeat Step 2 to Step 3 described in creating the first **Assign** activity to add the second **Assign** activity called 'SetPayload'.
3. Select the Copy Rules tab and expand the source and target trees:
 - In the From navigation tree, navigate to **Variable > Process > Variables > Invoke_ReadPayload_OutVariable > ZZ_CreateSingle_ShipDebitRequestProcessRequest**.
 - In the To navigation tree, navigate to **Variable > Process > Variables > Invoke_zzsdrequest_InputVariable > body**.

Drag the source node (ZZ_CreateSingle_ShipDebitRequestProcessRequest) to connect to the target node (body) that you just identified. This creates a line that connects the source and target nodes. The copy rule is displayed in the From and To sections at the bottom of the Edit Assign dialog box.

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 **BPEL Constructs** into the process diagram, between the second **Assign** activity and the **Invoke_zzsdrequest Invoke** activity.
2. Repeat Step 2 to Step 3 described in creating the first **Assign** activity to add the third **Assign** activity called 'SetRequestNumber'.
3. Select the Copy Rules tab and expand the source and target trees:
 - In the From navigation tree, navigate to **Variable > Process > Variables > inputVariable > Payload > client: ZZ_CreateSingle_ShipDebitRequestProcessRequest > client:input**.
 - In the To navigation tree, navigate to **Variable > Process > Variables > Invoke_zzsdrequest_InputVariable > Body > ns3:InputParameters > ns3: CP_SDR_HDR_REC > ns3:REQUEST_NUMBER** and select **ns3: TRX_NUMBER**.

Drag the source node (client:input) to connect to the target node (ns3: TRX_NUMBER) that you just identified. This creates a line that connects the source and target nodes. The copy rule is displayed in the From and To sections at the bottom of the Edit Assign dialog box.

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 **BPEL Constructs** into the process diagram, between the `Invoke_zzsrequest` **Invoke** and the **ReplyOutput** activities.
2. Repeat Step 2 to Step 3 described in creating the first **Assign** activity to add the fourth **Assign** activity called 'SetRequestNumber'.
3. Select the Copy Rules tab and expand the source and target trees:
 - In the From navigation tree, navigate to **Variable > Process > Variables > Invoke_zzsrequest_OutputVariable > body**.
 - In the To navigation tree, navigate to **Variable > Process > Variables > outputVariable > payload**.

Drag the source node (body) to connect to the target node (payload) that you just identified. This creates a line that connects the source and target nodes. The copy rule is displayed in the From and To sections at the bottom of the Edit Assign dialog box.

4. Click **OK** in the Assign window to complete the configuration of the **Assign** activity.

Configuring Web Service Policies

Use the following steps to add security policies at design time:

1. Navigate to SOA Content > Business Rules > composite.xml. Right click on the `ZZ_SDREQUEST` service and select "Configure WS Policies" from the drop-down list.
2. The Configure SOA WS Policies dialog appears.

In the Security section, click the **Add** icon (+). The Select Server Security Policies dialog appears.

Select 'oracle/wss_username_token_service_policy' and click **OK**.

The attached security policy is shown in the Security section.

3. From the navigation menu, select **View > Property Inspector** to display the Property Inspector window for `ZZ_SDREQUEST` service component.

In the Properties section, click the **Add** icon (+) for binding properties. The Create Property dialog appears.

Enter 'oracle.webservices.auth.username' in the Name field and enter 'operations' as the value.

Click **OK**.

4. Use the same approach by clicking the **Add** icon (+) again in the Properties section for binding properties. Enter 'oracle.webservices.auth.password' in the Name field. Enter the associated password for user 'operations' in the Value field.

Click **OK**.

Both selected property names and values appear in the Properties section.

Deploying and Testing the SOA Composite with BPEL Process at Runtime

To invoke the synchronous custom service (ZZ_SDREQUEST) from the BPEL client contained in the SOA composite, the SOA composite needs to be deployed to the Oracle WebLogic managed server. This can be achieved using Oracle JDeveloper. Once the composite is deployed, it can be tested from the Oracle Enterprise Manager Fusion Middleware Control Console.

Prerequisites

Before deploying the SOA composite with BPEL process using Oracle JDeveloper, you must have established the connectivity between the design-time environment and the runtime server. For information on how to configure the necessary server connection, see *Configuring Server Connection*, page B-1.

Note: If a local instance of the WebLogic Server is used, start the WebLogic Server by selecting Run > Start Server Instance from Oracle JDeveloper. Once the WebLogic Admin Server "DefaultServer" instance is successfully started, the <Server started in Running mode> and DefaultServer started message in the Running:DefaultServer and Messages logs should appear.

Perform the following runtime tasks:

1. Deploy the SOA Composite Application with BPEL Process, page 11-39
2. Test the SOA Composite Application with BPEL Process, page 11-40

Deploying the SOA Composite with BPEL Process

You must deploy the Create Single Supplier Ship and Debit Request BPEL process (ZZ_CreateSingle_ShipDebitRequest.bpel) contained in the SOA composite application that you created earlier before you can run it.

To deploy the SOA composite application:

1. In the Applications Navigator of JDeveloper, select the **ZZ_CreateSingle_ShipDebitRequest** project.

2. Right-click the project and select **Deploy** > [project name] > [serverConnection] from the menu.

For example, you can select **Deploy** > **ZZ_CreateSingle_ShipDebitRequest** > **SOAServer** to deploy the process if you have the connection set up appropriately.

Note: If this is the first time to set up the server connection, then the Deployment Action dialog appears. Select 'Deploy to Application Server' and click **Next**.

In the Deploy Configuration dialog, ensure the following information is selected before clicking **Next** to add a new application server:

- New Revision ID: 1.0
- Mark composite revision as default: Select this checkbox.
- Overwrite any existing composites with the same revision ID: Select this checkbox.

The steps to create a new Oracle WebLogic Server connection from Oracle JDeveloper are covered in *Configuring Server Connection*, page B-1.

3. In the Select Server dialog, select 'soa-server1' that you have established the server connection earlier. Click **Next**.
4. In the SOA Servers dialog, accept the default target SOA Server ('soa-server1') selection.
Click **Next** and **Finish**.
5. If you are deploying the composite for the first time from your Oracle JDeveloper session, the Authorization Request window appears. Enter the user name and corresponding password specified during the Oracle SOA Suite installation. Click **OK**.
6. Deployment processing starts. Monitor deployment process and check for successful compilation in the SOA - Log window.

Verify that the deployment is successful in the Deployment - Log window.

Testing the SOA Composite Application with BPEL Process

Once the BPEL process contained in the SOA composite application has been successfully deployed, you can manage and monitor the process from Oracle Enterprise Manager Fusion Middleware Control Console. For more information about Oracle SOA Suite, see the *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite*.

You can also test the process and the integration interface by manually initiating the process, and then 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 SOA composite application with BPEL process:

1. Navigate to Oracle Enterprise Manager Fusion Middleware Control Console (<http://<hostname>:<port>/em>). The login page appears.
2. Enter the user name and corresponding password specified during the installation. Click **Login** to log in to a farm. The composite (ShipNotice) you deployed is displayed in the Applications Navigation tree.

You may need to select an appropriate target instance farm if there are multiple target Oracle Enterprise Manager Fusion Middleware Control Console farms.

3. From the Farm navigation pane, expand the SOA >soa-infra node in the tree to navigate through the SOA Infrastructure home page and menu to access your deployed SOA composite applications running on soa-infra managed server. Click the ZZ_CreateSingle_ShipDebitRequest [1.0] link.
4. In the ZZ_CreateSingle_ShipDebitRequest [1.0] home page, click **Test**.
5. The Test Web Service page for initiating an instance appears. You can specify 'SD-Request1' as XML payload data to use in the Input Arguments section.

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.

Click **Test Web Service** to initiate the process.

The test results appear in the Response tab upon completion.

6. Verifying SOAP Response in the Console

In the Response tab page, click the Launch Message Flow Trace link to view the result of synchronous composite application. The Flow Trace page is displayed.

In the Trace section, verify that all components have a Completed state indicating that the application is processed successfully.

You can check the Faults section to see if any error occurred during the test.

7. Click your BPEL service component instance link (such as ZZ_CreateSingle_ShipDebitRequest) to display the Instances page where you can view the invocation details of the BPEL activities in the Audit Trail tab.

Click the Flow tab to check the BPEL process flow diagram. Click an activity of the

process diagram to view the activity details and flow of the payload through the process.

8. Double-click the `Invoke_ZZ_CreateSingle_ShipDebitRequest` icon from the process flow chart and click the **View XML document** link to open the XML file. This file records the Request ID that is returned for the transaction.
9. Log in to Oracle E-Business Suite as `trademgr` user and then select 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.
10. Verify if the request number 'SD-Request1' that you entered in Step 5 appears in the list.
11. Click the request number 'SD-Request' link. The Ship and Debit Request Details page is displayed allowing you to verify the request details.

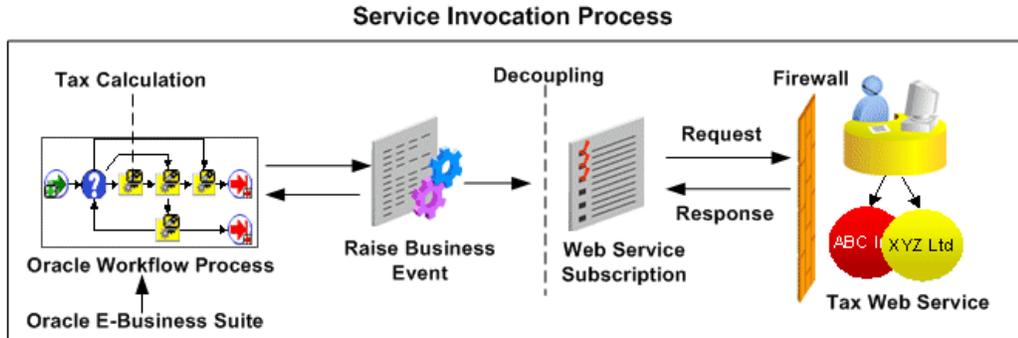
Using Service Invocation Framework to Invoke SOAP Services

SOAP Service Invocation Framework Overview

Oracle E-Business Suite Integrated SOA Gateway leverages Oracle Workflow Java Business Event System to provide infrastructure for SOAP and REST 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 events or services between systems, and implementing workflow process definitions at runtime as well as handling errors.

Since Oracle Workflow provides a total solution of managing and streamlining complex business processes and supporting highly-integrated workflow in Oracle E-Business Suite, Oracle E-Business Suite Integrated SOA Gateway relies on Oracle Workflow to enable the service invocation process. The following diagram illustrates the high level service invocation process flow:



In this diagram, when a tax calculation triggering event is raised from Oracle E-Business Suite through Oracle Workflow, the service that subscribes to this triggering event will be invoked to interact with an external tax system. If required, the external system will send a response message to acknowledge this request.

To be able to invoke external SOAP services from Oracle E-Business Suite, this invocation framework uses a wizard-based user interface in Oracle Workflow Business Event System to parse a given WSDL URL during the subscription creation and store the parsed service information or metadata as subscription parameters which will be used later at runtime during the actual service invocation.

Note: 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.

In summary, the SOAP service invocation framework provides the following functionality:

- It relies on the Business Event System to create events and event subscriptions and to parse a given WSDL representing a SOAP 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 SOAP 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 process synchronous and asynchronous subscriptions to the event.
- It utilizes the Error processing feature provided in the Business Event System to manage errors during subscription invocation and sends error notifications to the SYSADMIN user with the service definition, error and event details.
- It utilizes the workflow Notification System to send error notifications to and process responses from the SYSADMIN user.

For detailed information about Oracle Workflow, see the *Oracle Workflow User's Guide*, *Oracle Workflow Developer's Guide*, and the *Oracle Workflow Administrator's Guide*.

To better understand how service invocation framework is used in facilitating the SOAP service invocation, the following topics are discussed in this chapter:

Note: All outbound service invocation messages from Oracle E-Business Suite through service invocation framework are monitored using Service Invocation Monitor. See: *Monitoring and Managing Outbound Service Invocation Messages Using Service Invocation Monitor, Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

- Understanding SOAP Service Message Patterns, page 12-3
- Calling Back to Oracle E-Business Suite With SOAP Service Response, page 12-5
- Supporting SOAP Service Security, page 12-7
- Understanding SOAP Service Input Message Parts, page 12-7
- Understanding SOAP Service Invocation Metadata, page 12-12
- Managing SOAP Service Invocation Errors, page 12-15
- Defining SOAP Service Invocation Metadata, page 12-16
- Invoking SOAP Services, page 12-25
- Testing SOAP Service Invocation, page 12-29
- An Example of Invoking a SOAP Service from a Workflow Process, page 12-33
- Troubleshooting SOAP Service Invocation Failure, page 12-35
- Extending Seeded Java Rule Function for SOAP Services, page 12-41
- Other SOAP Service Invocation Usage Considerations, page 12-48

Understanding SOAP Service Message Patterns

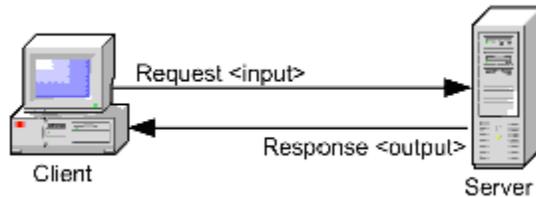
There are two major message exchange patterns — a request-response pattern, and a one-way (request - only) pattern.

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 processed as follows:

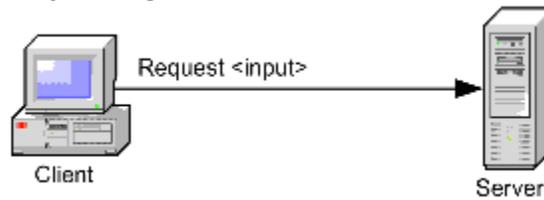
- A customer (the client) sends a request for an account balance to the Account Record Storage System (the server).
- The Account Record Storage System (the server) sends a reply to the customer (the client), specifying the dollar amount in the designated account.
- An *asynchronous* operation is one 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 processing the task until the response is ready.

Important: In this release, the SOAP 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, a client's zip code location sends updated weather data to the service when the local conditions change using the *request only* operation. The server updates the data but no response is sent back.

Calling Back to Oracle E-Business Suite With SOAP Service Response

To support synchronous request - response service operation, if a web service has an output or a response message, the SOAP 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 a web service operation where a client asks a service provider a question and then waits for a response before continuing on. For more information about this operation pattern, see *Understand SOAP Service 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 a "Raise" event activity and waits for a web service response using a "Receive" event activity, the invoker event key should be the 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* event and agent, the SOAP service invocation can be integrated back with a waiting workflow process or any other module within Oracle E-Business Suite. A SOAP 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 SOAP service with the service output message as the payload.

For example, it can be like:

```
WFBES_CALLBACK_EVENT=oracle.apps.wf.my.service.callback
```

- WFBES_CALLBACK_AGENT

This parameter can have a valid business event system agent to which the event with the service response message as the 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 instance, it can be like the default inbound agent (or any other inbound queue) for SOAP service messages:

```
WFBES_CALLBACK_AGENT=WF_WS_JMS_IN
```

Note: If you have defined custom agents, you can also specify the custom agent names as the parameter values.

Since the 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 a specialized agent listener on the callback agent to process the callback event.

For example, if the callback event for a service invocation is `oracle.apps.wf.myervice.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.myervice.callback` is used as Correlation ID in `WF_WS_JMS_IN` to help create an agent listener to process that event.

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 subscription parameters. For instance, the event parameters are passed as follows:

- `BusinessEvent.setStringProperty("WFBES_CALLBACK_EVENT", "oracle.apps.wf.myervice.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 12-22.

The better understand how to invoke a web service, see An Example of Invoking a SOAP Service from a Workflow Process, page 12-33

Supporting SOAP Service Security

The SOAP service invocation framework supports WS-Security through **UsernameToken based security**.

This security mechanism authenticates the user invoking a SOAP service by passing a *user name* and an optional *password* in the SOAP Header of a SOAP request sent to the web service provider.

When creating the event subscription for the SOAP service to be invoked, the developer needs to specify the user name and password information through the design-time event subscription user interface.

Note that the SOAP service security is also handled based on the customization level. The customization level is used to protect Oracle E-Business Suite seed data and to preserve your customizations in an upgrade. If the Invoke Web Service event subscription's customization level is Core or Limit, and if the user name is supplied by the subscription owner, it cannot be updated. If the user name is not already supplied, you can update the user name if it's required. The Password field can always be updated regardless of the customization level if it is required.

For more information about UsernameToken based security and customization level usage, see Understanding and Configuring WS-Security for the SOAP Service Invocation Framework, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

Understanding SOAP 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 part and a body part in the input message.

For example, consider the operation PROCESSPO in Oracle E-Business Suite XML Gateway service (`http://<host>:<port>/webservices/SOAPProvider/xmlgateway/ont__poi/?wsdl`) as described below.

```

<definitions targetNamespace="ONT__POI" targetNamespace="http://xmlns.
oracle.com/apps/ont/soapprovider/xmlgateway/ont__poi/">
<type>
  <schema elementFormDefault="qualified" targetNamespace="http://xmlns.
oracle.com/apps/ont/soapprovider/xmlgateway/ont__poi/">
    <include schemaLocation="http://<hostname>:
<port>/webservices/SOAPProvider/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/SOAPProvider/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, this section includes the following topics:

- Event Payload as SOAP Body, page 12-8
- Other Web Service Input Message Parts, page 12-11

Event Payload as SOAP Body

Any detailed 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 the *Oracle Workflow Developer's Guide* and the *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 a 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 a 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 the 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 the 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.xml 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 an event payload, the SOAP 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 the same as the part name in the input message definition in WSDL.

For example, the header part for the above example is passed to the 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 user name, responsibility, responsibility application, and NLS language elements for web service authorization:

```
String headerPartMsg = "<SOAHeader
xmlns:=\"http://xmlns.oracle.com/xdb/SYSTEM\" " +
    "env:mustUnderstand=\"0\"
xmlns:env=\"http://schemas.xmlsoap.org/soap/envelope/\"> \n" +
    " <MESSAGE_TYPE>XML<MESSAGE_TYPE>\n" +
    " <MESSAGE_STANDARD>OAG<MESSAGE_STANDARD>\n" +
    " <TRANSACTION_TYPE>PO<TRANSACTION_TYPE>\n" +
    " <TRANSACTION_SUBTYPE>PROCESS<TRANSACTION_SUBTYPE>\n" +
    " <DOCUMENT_NUMBER>xxx<DOCUMENT_NUMBER>\n" +
    " <PARTY_SITE_ID>xxxx<PARTY_SITE_ID>\n" +
    "<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 interface `oracle.apps.fnd.wf.bes.InvokerConstants` for use in Java code.

If the SOAP service input message definition has several parts, value for the part that is sent as SOAP body is passed as an 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);
```

Understanding SOAP Service Invocation Metadata

To invoke a SOAP service through Business Event System, you must create a subscription to the event that triggers the invocation. Before creating an event subscription with all the data required for the invocation, you should first understand the concepts of invocation parameters and metadata and how they work behind the scene to invoke a SOAP service.

This section describes the required parameters that you need to provide during the event subscription creation through the following topics:

- Invoke Web Service Wizard, page 12-12
- SOAP Service Security, page 12-14
- Additional Subscription Parameters, page 12-14

Invoke Web Service Wizard

To subscribe to a SOAP service 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 SOAP service.

With the "Invoke Web Service" action type, the Create Event Subscription - Invoke Web Service wizard appears where you need to provide a WSDL URL representing the SOAP service to be invoked. The SOAP service can be of any type, such as a native service or BPEL process.

Note: A BPEL process itself is a 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 Oracle SOA Suite to connect to services.

When a triggering event occurs, the Business Event System processes the subscription through the seeded Java function and invokes the BPEL process.

Create Event Subscription - Invoke Web Service Wizard

The screenshot shows a wizard interface with five steps: Load WSDL, Select Service, Select Service Port, Select Operation, and Configuration. The 'Load WSDL' step is currently active. The main content area is titled 'Select a WSDL Source' and includes a 'Cancel' button, 'Step 1 of 5', and a 'Next' button. Below the title, there are several personalization options: 'Personalize Flow Layout: (LoadWSDLFN)', 'Personalize Stack Layout: (LoadWSDLRN)', 'Personalize "Required Field Description"', and 'Personalize Table Layout: (LoadTableRN)'. A note indicates that an asterisk (*) denotes a required field. A text input field is labeled '* WSDL URL' and contains an example URL: 'http://supplier.company.com:8888/webservices/supplier_service.wsdl'.

The entered WSDL information will be parsed into service metadata for your further selections later in the wizard. The service metadata you selected along with the WSDL URL will be stored as the following subscription parameters and displayed in the Web Service Details region:

- SERVICE_WSDL_URL
- SERVICE_NAME
- SERVICE_PORT
- SERVICE_PORTTYPE
- SERVICE_OPERATION

Note: The SOAP address derived from the SERVICE_WSDL_URL parameter captured here during the event subscription can be overridden by an event parameter called SERVICE_SOAP_ADDRESS if passed at runtime during service invocation.

The seeded Java Rule Function for SOAP services `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 SOAP Service Seeded Rule Function*, page 12-41.

SOAP Service Security

Configuring Web Service Security for SOAP Services

If the SOAP service being invoked enforces Username/Password based authentication, then this framework also supports the UsernameToken based WS-Security header during the service invocation.

Important: This UsernameToken based WS-security header is implemented during the service invocation only if the web service provider that processes the web service request needs this security header.

After entering needed information in the Create Event Subscription - Invoke Web Service wizard, the Web Service Security region is displayed letting you specify or update the user name and password if appropriate. The information will then be stored in Vault securely.

Oracle Workflow allows various levels of updates on business event and subscription based on the customization level. If the Invoke Web Service event subscription's customization level is Core or Limit, and if the user name is supplied by the subscription owner, it cannot be updated. If the user name is not already supplied, you can update the user name if required. The Password field can always be updated regardless of the customization level if it is required.

For more information about UsernameToken based security and customization level usage, see *Understanding and Configuring WS-Security for the SOAP Service Invocation Framework, Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

Additional Subscription Parameters

Setting Additional Subscription Parameters

Apart from the subscription parameters that have been parsed and stored through the Invoke Web Service Subscription page, the following information could 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 the service invocation:

- WS-Security Header

To help protect the security header `<wsse:Security>` from being reused during a replay attack, you can optionally use the following parameter to set the expiration time for the header:

- WFBES_SOAP_EXPIRY_DURATION

By default, the security header is set to expire 60 seconds in the `<wsu:Timestamp>` element after it is created. This parameter provides an option letting you to set a

different expiration time in seconds for the header.

- **Message transformation**

If the invoker event's XML payload (to be used as a SOAP 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 service. Similarly, if the 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 SOAP 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 SOAP Service Input Message Parts*, page 12-7.

- **Callback: Callback to Oracle E-Business Suite with web service response**

- WFBES_CALLBACK_EVENT
- WFBES_CALLBACK_AGENT

To process a SOAP 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 launched.

For more information on these callback parameters, see *Calling Back to Oracle E-Business Suite With SOAP Service Response*, page 12-5.

Managing SOAP Service Invocation Errors

The SOAP service invocation framework uses the same way of handling errors in the Business Event System to manage errors if occur during the implementation of business event subscriptions. If the service invocation returns a fault message, the event is enqueued to an error queue to trigger error processing. If an exception occurred during the 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, the SOAP 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 itemtype.

For example, if there is a runtime exception when the Workflow Java Deferred Agent Listener processes an event subscription to invoke a 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 processes the error subscription which sends a notification to SYSADMIN with the web service definition, error details, and event details. Since Oracle Workflow default event error handler provides options for SYSADMIN to retry the service invocation process after verifying that the reported error has been corrected, SYSADMIN can invoke the service again from the notification if necessary.

However, if there is a runtime exception when invoking the service by raising the Invoker event with synchronous subscription (phase less than 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

To access the Create Event Subscription page, log in to Oracle E-Business Suite as a user who has the Workflow Administrator Web Applications responsibility. Select **Business Events** from the navigation menu and choose the **Subscriptions** subtab. In the Event Subscriptions page, click **Create Subscription**.

For detailed information on how to create an error subscription for service invocation, see Create an Error subscription with 'Launch Workflow' Action Type, page 12-21.

Defining SOAP Service Invocation Metadata

Because the service invocation is taken place in the Oracle Workflow Business Event System, before invoking a SOAP service, the 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 SOAP Service Invoker Business Event, page 12-17

A SOAP service invoker business event that serves as a request message (or a SOAP service input message) for a service needs to be created first.

2. Creating a Local Subscription to the SOAP Service Invoker Event, page 12-19

This event subscription indicates that when a triggering event occurs, the action item of this subscription is to invoke a SOAP service defined as part of this subscription.

3. Creating an Error Subscription to Enable Error Processing for the SOAP Service, page 12-21

This error subscription enables error processing in the Business Event System that is used to communicate with the SYSADMIN user of an error condition in subscription processing.

4. Creating a Receive Event for SOAP Response (Optional), page 12-22

This step is required only if a SOAP service has an output or a response message to communicate or call back to Oracle E-Business Suite.

If a SOAP service does not require a response, then you do not need to create a receive event.

5. Creating a Receive Event Subscription for SOAP Response (Optional), page 12-24

Once a receive event is in place, you must create an External subscription to the receive event to pass the SOAP service response message.

Step 1: Creating a SOAP 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 SOAP service invoker business event.

The invoker event can be served as a request message (or a web service input message) in a message pattern to send inquiries to a service.

To invoke a SOAP service through the Business Event System, you must create an invoker business event, and then subscribe to the SOAP service invoker event later with an appropriate action type.

Note: In this release, the SOAP service invocation framework supports the following types of service invocation:

- **One-way (request only)** - A consumer or client sends a message to a service, and the service does not need to reply.
- **Synchronous request-response** - This type 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 call back into Oracle E-Business Suite after a SOAP service is successfully invoked. See *Creating a Receive Event for SOAP Response (Optional)*, page 12-22 and *Creating a Receive Event Subscription for SOAP Response (Optional)*, page 12-24.

For more information about business events, see *Events, Oracle Workflow Developer's Guide*.

To create an invoker event:

1. Log in to Oracle E-Business Suite as a user who has 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.xxx.user.webservice.invoke`
 - Display Name: Enter an event display name, such as `oracle.apps.xxx.user.webservice.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')
4. Click **Apply** to save your work.

For more information on how to create a business event, see the *Oracle Workflow Developer's Guide* for details.

Step 2: Creating a Local Subscription to the SOAP Service Invoker Event

After creating a SOAP service invoker event in step 1, you need to create a local subscription to the invoker event with "Invoke Web Service" action type. When a triggering event occurs, the Business Event System processes the subscription through the seeded Java function and invokes the SOAP service as indicated by the action type.

To create a local event subscription with 'Invoke Web Service' action type:

1. Log in to Oracle E-Business Suite as a user who has 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.xxx.user.webservice.invoke`
 - Phase: 90

If the event is raised from Java, the phase number determines whether an event will be invoked right away or enqueued to the WF_JAVA_DEFERRED queue.

Note: If the invoker event is raised from PL/SQL, it is always deferred to the WF_JAVA_DEFERRED queue regardless of the phase because the subscription has a Java rule function that cannot be processed in the database.

- If the phase is greater than or equal to 100, then the event is enqueued to the WF_JAVA_DEFERRED queue and will be dispatched later.
- If the phase is less than 100, then the event is dispatched immediately to the

Java Business Event System soon after a 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.
 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. Click **Next** to display all ports for a selected service
 3. Select an appropriate service port and click **Next** to display all operations for a selected port.
 4. Select an appropriate service operation and click **Next**. This opens the last page of the Create Event Subscription - Invoke Web Service wizard.
 5. All the selected service metadata information is automatically displayed in the Web Service Details region.
 6. In the Web Service Security region, enter information in the Username and Password fields if appropriate.
 7. In the Web Service Invoker region, the default Java Rule Function for SOAP services `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.

8. In the Documentation region, enter an application name or a program name that owns the subscription (such as 'Oracle Workflow') in the Owner Name field. Enter 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*.

Step 3: Creating an Error Event Subscriptions to Enable Error Processing for the SOAP Service

To enable the error processing feature during the service invocation, you must create an Error subscription with "Launch Workflow" action type to the SOAP invoker event.

Once subscribing to this error processing, if any errors occurred during the invocation, the error process sends a workflow notification to the SYSADMIN user. This notification includes the SOAP service definition, event details, and error details allowing the SYSADMIN user to easily identify the error. The notification also provides an option for the SYSADMIN user to respond to the error. The SYSADMIN user can invoke the SOAP 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 SOAP Service Invocation Errors*, page 12-15.

To create an error subscription with 'Launch Workflow' action type:

1. Log in to Oracle E-Business Suite as a user who has 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.xxx.user.webservice.invoke`
 - Phase: This can be any phase number.
 - Status: Enabled

Note: While updating an event and an event subscription, for seeded events with a customization level of Limit, you can only update the status. For seeded product-specific events with a customization level of Core, you cannot update any properties. You can only view the subscription definition.

For information on how to use customization level, see *Configuring Security Password with Customization Level*,

- Rule Data: Key
 - Action Type: Launch Workflow
 - On Error: Stop and Rollback
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 a 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.
 7. Click **Apply**.

Step 4: Creating a Receive Event for SOAP Response (Optional)

A receive event can serve as a communication vehicle to communicate or call back to Oracle E-Business Suite if a SOAP service has an output or response message required to be communicated back after the service has been successfully invoked. However, whether you need to create a receive event and an 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 Service Invocation

- If the SOAP service invoker event is raised from Java code in the application tier, and the SOAP service invoker subscription is synchronous with the subscription phase less than 100, then the service is invoked as soon as the event is raised. If the invocation is 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 a callback event. Hence, you may not need to create a receive event.

- If the service invoker event is raised from Java code with the subscription phase greater than or equal to 100, or if the event is raised from PL/SQL, the event message will be enqueued to the `WF_JAVA_DEFERRED` queue. In this situation, you will need to create a receive event to the event if the service has an output or a response message. A 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 service response is available.

See: Calling Back to Oracle E-Business Suite With SOAP Service Response, page 12-5.

For Request-only Web Service

If it is a request-only SOAP 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.xxx.user.webservice.receive`
 - Display Name: Enter an event display name, such as `oracle.apps.xxx.user.webservice.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')
3. Click **Apply** to create a receive event.

Step 5: Creating a Receive Subscription for SOAP Response (Optional)

If you create a receive event as described in step 4, you must create an external event subscription to the receive event. The SOAP 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.

To create a receive event subscription:

1. Log in to Oracle E-Business Suite as a user who has 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.xxx.user.webservice.receive`
 - Phase: any phase number
 - Status: Enabled
 - Rule Data: Key
 - Action Type: any action type
 - On Error: Stop and Rollback

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 any other action type is selected, then a different type of the create event subscription page is displayed. By entering an appropriate action type through the subscription page, you can launch a workflow process or invoke 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 a workflow type that is waiting for the response
 - Workflow Process: Enter a workflow process that is waiting for the response
 - Priority: Normal
6. In the Documentation region, enter an application or a program name in the Owner Name field (such as 'Oracle Workflow'). Enter an application or a program ID in the Owner Tag field (such as 'FND').
 7. Click **Apply**.

Invoking SOAP 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 a PL/SQL or Java layer.

Service Invocation from PL/SQL

1. An 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 the *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 the SOAP service Java Rule Function `oracle.apps.fnd.wf.bes.WebServiceInvokerSubscription`.
3. The Business Event System enqueues the event message to the `WF_JAVA_DEFERRED` queue. The Java Deferred Agent Listener then dequeues and processes the subscription whose Java rule function invokes the SOAP service.
4. If callback event and agent parameters are mentioned, the service response is communicated back to Oracle E-Business Suite using the callback information. The Java Deferred Agent Listener process that runs on the Concurrent Manager (CM) tier invokes the SOAP service.

Service Invocation from Java

1. A Java application raises a business event using Java method `oracle.apps.fnd.wf.bes.BusinessEvent.raise` either from an OA Framework page

controller/AMImpl or Java code running on the 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 processed inline or deferred is determined by the phase of the subscription.
 - If the invoker subscription is created with the Phase value greater than or equal to 100, the event is enqueued to the `WF_JAVA_DEFERRED` queue.
 - If the invoker subscription is created with the Phase value less than 100, the event is dispatched inline.

If the event is raised from an OA Framework page, the dispatch logic runs within `OACORE WebLogic Server`.

Note: If the SOAP service invoker event is raised from Java on the application tier, and the invoker subscription is synchronous with the subscription phase value less than 100, then the SOAP service is invoked as soon as the event is raised. If the invocation is successful, the response can be read by the calling application and is available immediately by using method `BusinessEvent.getResponseData()`.

`oracle.apps.fnd.wf.bes.BusinessEvent.raise` throws `oracle.apps.fnd.wf.bes.BusinessEventException` if there are any issues while invoking a SOAP service inline. `BusinessEventException` object internally stores the underlying root cause exception within a `LinkedException` object. In order to see the complete exception details, print the exception stack trace from `BusinessEventException.getLinkedException()`.

If the event is raised from Java with the subscription phase value greater than or equal to 100 or if the event is raised from PL/SQL, the event message will be enqueued to the `WF_JAVA_DEFERRED` queue. If the SOAP service has an output or a response message, a 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 a SOAP 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 {
            // Proxy 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 =
"<IntegrationRepositoryService_GetInterfaceFunctionByName
xmlns:=\"http://xmlns.oracle.com/apps/fnd/rep/ws\"> \n"+
<fullName>SERVICEBEAN:/oracle/apps/fnd/rep/ws/IntegrationRepos
itoryService:getInterfaceFunctionByNameSERVICEBEAN:/oracle/apps/fnd/
rep/ws/IntegrationRepositoryService:
getInterfaceFunctionByName</fullName>\n"+
<IntegrationRepositoryService_GetInterfaceFunctionByName>;
            event.setData(input);

            String headerPartMsg = "<SOAHeader
xmlns:=\"http://xmlns.oracle.com/xdm/SYSTEM\" " +
"env:mustUnderstand=\"0\"
xmlns:env=\"http://schemas.xmlsoap.org/soap/envelope/\"> \n" +
" <MESSAGE_TYPE>XML<MESSAGE_TYPE>\n" +
" <MESSAGE_STANDARD>OAG<MESSAGE_STANDARD>\n" +
" <TRANSACTION_TYPE>PO<TRANSACTION_TYPE>\n" +
" <TRANSACTION_SUBTYPE>PROCESS<TRANSACTION_SUBTYPE>\n" +
" <PARTY_SITE_ID>xxx<PARTY_SITE_ID>\n" +
" <DOCUMENT_NUMBER>xxx<DOCUMENT_NUMBER>\n" +
" <SOAHeader>\n"
businessEvent.setStringProperty("WFBES_INPUT_header",
headerPartMsg);
            event.raise(conn);
            conn.commit();

            Object resp = event.getResponseData();
            if (resp != null) {

```

```

String respStr = resp.toString();
    // Process web service response here
    }
    else {
        // Either web service invocation failed and no exception was
        thrown
        // or the web service is one-way or asynchronous and
        did not return
        // a valid response
        System.out.println("No response received");
    }
}
catch (BusinessEventException e) {
    // Use appropriate logging mechanism as per your coding
    standards
    // instead of System.out.println
    System.out.println("Exception occurred " + e.
getLinkedException().getMessage());
    // Print the complete exception stack to log file for
    troubleshooting
    // Most importantly, if an exception occurred, do not proceed to
    process the
    // response
    e.getLinkedException().printStackTrace();
}
catch (Exception e) {
    // Use appropriate logging mechanism as per your coding
    standards
    // instead of System.out.println
    System.out.println("Exception occurred " + e.getMessage());
    // Print the complete exception stack to log file for
    troubleshooting
    // Most importantly, if an exception occurred, do not proceed to
    process the
    // response
    e.printStackTrace();
}
}
}

```

Important: When invoking a SOAP service using the SOAP service invocation framework, the invoker business event is raised using the `oracle.apps.fnd.wf.bes.BusinessEvent.raise (Connection)` method that requires a JDBC connection to be passed.

To get the JDBC connection, always use the current application context object available for your scenario. For example, if a service invocation is from an OA Framework page, then get the JDBC connection from the `OAPageContext` object. If it is from a concurrent program, get the JDBC connection from the `CpContext` object. You should not create a `WorkflowContext` in these situations. Otherwise, a duplicate application context will be unnecessarily created. A new `WorkflowContext` should be created only if JDBC connection is not already available through other means.

Testing SOAP Service Invocation

The SOAP service invocation framework uses the Oracle Workflow Test Business Event page to check the basic operation of the Business Event System by raising a test event from either a Java or PL/SQL layer and by processing synchronous or asynchronous subscriptions to that event. This testing feature lets you easily validate whether a SOAP service can be successfully invoked from the concurrent manager tier and OACORE WebLogic Server.

You can test a service invocation using one of the following ways:

- Using the Test Business Event Page to Manually Raise an Event, page 12-29
- Using Command Line to Raise an Event, page 12-32

Using the Test Business Event Page

Use the Test Business Event page to test an event by raising it from PL/SQL API or Java.

- For an invoker event raised using the **Raise in Java** option, the SOAP service is invoked from the OACORE WebLogic server if the subscription phase is less than 100.

If the 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 service using synchronous subscription, the exception message is shown on the Test Business Event page.

- For an invoker event raised using the **Raise in PLSQL** option, the service is invoked from the concurrent manager tier. The raised event will be enqueued to WF_JAVA_DEFERRED and then dispatched by the Workflow Java Deferred Agent Listener.

The seeded Java rule function uses the callback event and agent to communicate the response or service output message back to Oracle E-Business Suite through Business Event System.

Note: Since the Workflow Java Deferred Agent Listener is responsible for dispatching the subscription and invoking services from the concurrent manager tier, ensure that the 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 the Workflow Java Deferred Agent Listener to view its status.

Testing Service Invocations

After logging in to Oracle E-Business Suite as a user who has the Workflow Administrator Web responsibility, select **Business Events** from the navigation menu. 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 a 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 the XML response message if appropriate.

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 the data that you have entered.

Following parameters may be specified when raising the event from the Test Business Event page to invoke a SOAP service:

- Message transformation: XSL transformation for SOAP service input messages and output messages.
 - WFBES_OUT_XSL_FILENAME
 - WFBES_IN_XSL_FILENAME
- 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 the event raise.

- Runtime SOAP service address: If passed at runtime, it overrides the SOAP service endpoint derived from the subscription parameter SERVICE_WSDL_URL.
 - SERVICE_SOAP_ADDRESS
- Callback: Callback to Oracle E-Business Suite with SOAP service response.
 - WFBES_CALLBACK_EVENT
 - WFBES_CALLBACK_AGENT
- SOAP Body:
 - XML Input message (Required)
- WS-Security: Information required to add UsernameToken header to a SOAP request.

The SOAP service security information is entered in the Web Service Security region of the event subscription page after entering needed information in the Invoke WSDL wizard. See: Creating a Local Subscription to the SOAP Service Invoker Event, page 12-19.

Note: As described here that security information is now entered through the event subscription user interface to replace the security parameters used earlier in Oracle E-Business Suite Release 12.1.

These WS-Security parameters (WFBES_SOAP_USERNAME, WFBES_SOAP_PASSWORD_MOD, and WFBES_SOAP_PASSWORD_KEY) are now maintained internally by the framework.

For information about these parameters, see:

- Understanding SOAP Service Input Message Parts, page 12-7
- Supporting SOAP Service Security, page 12-7
- Calling Back to Oracle E-Business Suite With SOAP Service Response, page 12-5

Testing Invocation with Callback Required

If you want to test an invocation with callback to Oracle E-Business Suite, 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)

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.

To test an event invocation:

1. Log in to Oracle E-Business Suite as a user who has the Workflow Administrator Web responsibility and select **Business Events**.
2. Search on a business event that you want to run the test, such as `oracle.apps.xxx.user.webservice.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 an input message. For example, you can enter:

```
<Process xmlns="http://xmlns.oracle.com/MyFirstSOAComposite_jws/UseMovieService/InvokeMovie">
  <zipCode>32822</zipCode>
  <radius>5</radius>
</Process>
```

7. Click **Raise in Java** to raise an event from the OACORE WebLogic server.

If the SOAP service is successfully invoked, a confirmation message appears on top of the Test Business Event page indicating that the event (`oracle.apps.xxx.user.webservice.invoke`) has been successfully raised.

This test page reloads and displays the XML Response region right after the XML Content field.
8. Click **Raise in PLSQL** to raise an event from the concurrent manager tier.
9. If errors occur, the Event Error Details region appears letting you view the error details.

Viewing Error Details in the Test Business Event Page

If any errors occur while testing the service invocation, an error message appears indicating that errors occurred while dispatching the event and detailed information is shown in the Event Error Details region.

The Event Error Details region lets you quickly view error information through the same page for easier debugging.

Optionally, to see detailed log messages that capture each occurrence in sequential order for service invocation, before testing the invocation, 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 SOAP Service Invocation Failures on OACORE WebLogic Server*, page 12-36.

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 PL/SQL or Java based events.

- For PL/SQL based events, use the PL/SQL `WF_EVENT.Raise` API to test the SOAP service invocation from the concurrent manager tier JVM. The Workflow Java Deferred Agent Listener dispatches the subscription and invokes the services from the concurrent manager tier.

Note: Since the Workflow Java Deferred Agent Listener is responsible for dispatching the subscription and invoking the services from the concurrent manager tier, ensure that the 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 the Workflow Java Deferred Agent Listener to view its status.

- For Java-based web events, use the Java method `oracle.apps.fnd.wf.bes.BusinessEvent.raise` to test the service invocation.

For example, we could have a test class `oracle.apps.fnd.wf.bes.WFInvokerTestCase` with classpath set to `$AF_CLASSPATH`.

```
java oracle.apps.fnd.wf.bes.WFInvokerTestCase <DBC file> <proxy
host> <proxy port>
```

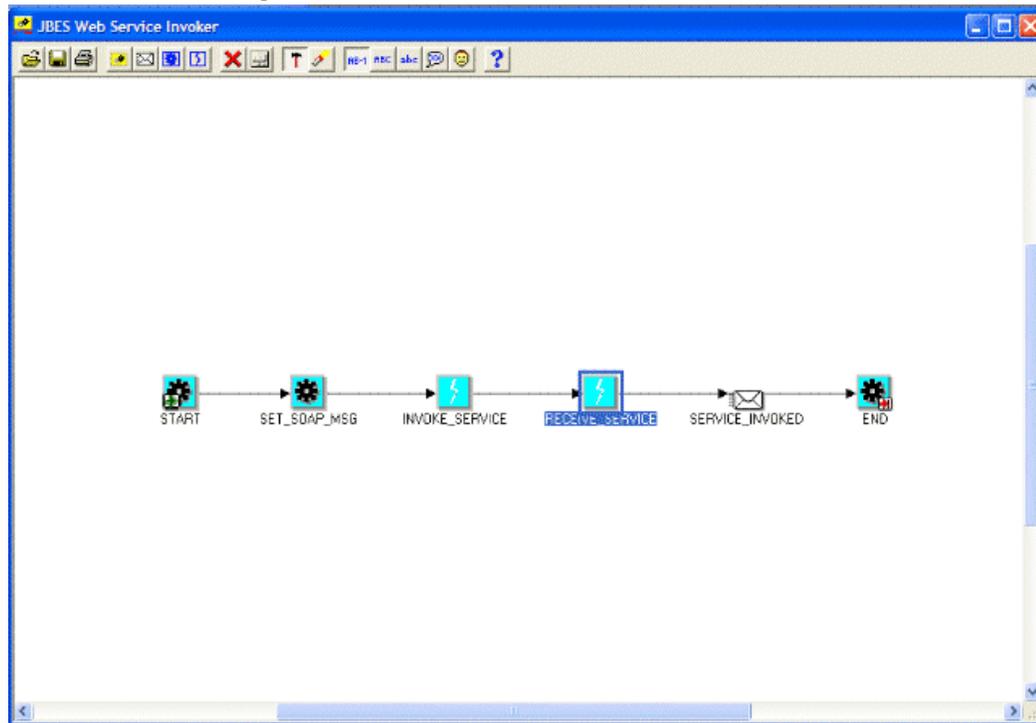
An Example of Invoking a SOAP Service from a Workflow Process

The following example is to invoke a SOAP service through launching a workflow process including the following nodes or activities:

- An invoker business event to invoke a SOAP service.
For example, `INVOKE_SERVICE` is an event activity with event action "Raise".
- A receive business event to receive a response or 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 the service invocation response, regular transaction processing and so on.
For example, `SERVICE_INVOKED` is a notification activity to send a notification message when a SOAP service is successfully invoked.

The following workflow process diagram illustrates the service invocation process flow:

Workflow Process Diagram to Invoke a SOAP Service



Defining Service Invocation Metadata

To define the service invocation metadata with the callback feature, you must have the following necessary events and subscriptions in place:

1. An invoker event, such as `INVOKE_SERVICE` in the workflow diagram.
This activity is used to pass the event XML payload as the SOAP body and other event parameters required for the web service invocation.
See: *Creating a Web Service Invoker Business Event*, page 12-17.
2. Local and error event subscriptions to the invoker event. See: *Creating Local and Error Event Subscriptions to the Invoker Event*, page 12-19.
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 it is for the 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 a successful web service invocation.

See: Creating a Receive Event and Event Subscription (Optional), page 12-22.

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 in to Oracle E-Business Suite as a user who has the Oracle Workflow Web Administrator responsibility.
2. From the navigation menu, select **Oracle Applications Manager > Workflow Manager**.
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.

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.

For more information on how to create and launch a workflow, see the *Oracle Workflow Developer's Guide*.

Troubleshooting SOAP Service Invocation Failure

SOAP services can be invoked from any one of following tiers:

- **OACORE WebLogic Server:** SOAP service invocations from OA Framework page using a synchronous event subscription (phase is less than 100) are processed from within the OACORE WebLogic Server.
- **Concurrent Manager (CM) Tier JVM:** The following SOAP service invocations are processed from CM tier JVM:
 - By 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
- By Java Concurrent Program
 - Invocations performed directly from within a Java Concurrent Program.
- **Standalone JVM:** SOAP service invocations from a Java process that runs outside OACORE or CM using a synchronous event subscription are processed from within that JVM.

In most cases, the SOAP service to be invoked resides outside the firewall and the running host does not have direct access to the WSDL or the service endpoint to send the SOAP request. Without properly setting up and configuring the proxy parameters for each tier that the service invocation may occur, WSDL files will not be parsed and consumed during the subscription or the services will not be successfully invoked.

For information on how to set up proxy host and port appropriately at each layer, see detailed information described in the Setup Tasks, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

At runtime, if a SOAP service invocation fails, an exception is thrown and the invoker event is enqueued to the WF_ERROR queue. Since the service can be invoked from any of the three tiers, this section provides the troubleshooting information to help you resolve the failure invocation based on the tier that service invocation could occur:

- Troubleshooting SOAP Service Invocation Failure on OACORE WebLogic Server, page 12-36
- Troubleshooting SOAP Service Invocation Failure on Concurrent Manager (CM) Tier JVM, page 12-39
- Troubleshooting SOAP Service Invocation Failure on Standalone JVM, page 12-41

Troubleshooting SOAP Service Invocation Failure on OACORE WebLogic Server

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 running 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:

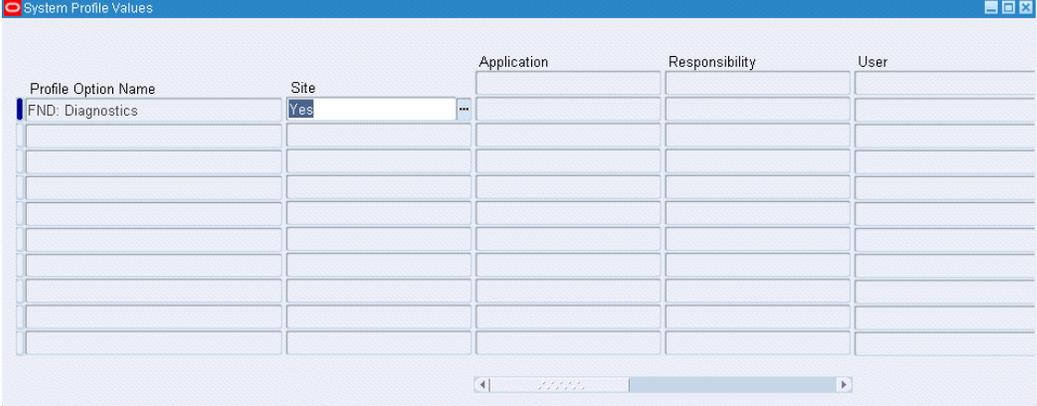
1. Setting FND: Diagnostics Profile Option, page 12-37
2. Displaying On-screen Logging, page 12-37

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 Form for Setting FND: Diagnostics Profile Option



The screenshot shows a web browser window titled "System Profile Values". The main content is a table with the following columns: Profile Option Name, Site, Application, Responsibility, and User. The first row contains the value "FND: Diagnostics" in the Profile Option Name column and "Yes" in the Site column. The rest of the table is empty. A scrollbar is visible at the bottom of the table.

Profile Option Name	Site	Application	Responsibility	User
FND: Diagnostics	Yes			

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, you enable the on-screen logging feature. 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 12-37.

To display on-screen logs while testing your service invocation in the Test Business Event page:

1. Log in to Oracle E-Business Suite as a user who has the Workflow Administrator Web responsibility. Select the Business Events link to locate an invoker business event that you want to run the test, such as `oracle.apps.xxx.user.webservice.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.
6. Navigate to the Test Business Event page and raise an event to run 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 invocation status, exception or fault if there is any, and whether the callback succeeded or not.

The following example log indicates that the service invocation 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, corr
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 12-36.

Troubleshooting SOAP Service Invocation Failure on Concurrent Manager (CM) Tier JVM

To troubleshoot SOAP service invocation failure on the Concurrent Manager (CM) Tier JVM, you must ensure that the Error subscription is created for the all SOAP service invoker events to capture complete exception details when the invocation happens from Workflow Java Deferred Agent Listener.

Error Subscription

For all SOAP 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 processing. It sends a workflow notification to SYSADMIN with the SOAP 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 WFEERROR:DEFAULT_EVENT_ERROR2, the Workflow Java Error Agent Listener processes the error subscription which sends a notification to SYSADMIN with the SOAP service definition, error details, and event details.

For more information, see *Managing SOAP Service Errors*, page 12-15.

Enabling Logging for Workflow Java Deferred Agent Listener

Since Oracle Workflow default event error handler provides options for SYSADMIN to retry the service invocation process after verifying that the reported error has been corrected, SYSADMIN can invoke the service again from the notification if necessary. However, if further analysis of the steps leading to the exception is required, use the Workflow Java Deferred Agent Listener logging mechanism to set the STATEMENT level log for Workflow Java Deferred Agent Listener and retry the failed service invocation to obtain detailed steps leading to the exception.

For more information, see *Java Agent Listeners, Oracle Workflow Administrator's Guide*.

Enabling Logging for Java Concurrent Program

To enable logging for the SOAP service invocation framework when used from within a Java Concurrent Program to invoke SOAP services, set the following profile option values for the right application context based on how JDBC connection is obtained to

raise the business event:

- FND: Debug Log Enabled (AFLOG_ENABLED) profile value set to 'Yes'
- FND: Debug Log Level (AFLOG_LEVEL) profile value set to 'Statement'
- FND: Debug Log Module (AFLOG_MODULE) profile value set to '%fnd.wf.bes%, %fnd.sif%'

This profile value can be a set, as comma separated module names such as "fnd.wf.bes%, fnd.sif%" as one string without quotes. If you would like to enable log for all modules, set it to %.

Oracle strongly recommends that you get the JDBC connection directly from the concurrent program context itself as listed below, then use the application context (User and Responsibility) under which the Java Concurrent Program request was submitted. Using the JDBC connection from the concurrent program context helps to group all logs for that concurrent program request and the service invocation framework invocation under that Request ID. After the concurrent program completes the processing, use the Request ID to search for all the log messages for the component 'Concurrent Program'.

```
Connection conn = cpContext.getJDBCConnection();
event.raise(conn)
```

However, if you use an unrecommended approach by creating a separate WorkflowContext to get the JDBC connection from it, as listed below:

```
WorkflowContext wfCtx = new WorkflowContext();
Connection conn = wfCtx.getJDBCConnection();
event.raise(conn)
```

Use the following context to set the logging profile options:

- User: SYSADMIN
- Responsibility: System Administrator (SYSTEM_ADMINISTRATOR)

Although the Java Concurrent Program would be processed under the application context under which the request was submitted, creating a new WorkflowContext within the program results in setting a hard-coded application context of SYSADMIN and SYSTEM_ADMINISTRATOR for its JDBC connection. Hence, this approach is not recommended. A new WorkflowContext should be created only if the JDBC connection is not already available through other means.

Since there are two separate application contexts, one created for concurrent program context and a new one for WorkflowContext, the logs for the concurrent program request and the invocation through the SOAP service invocation framework are stored under the two different contexts.

After the concurrent program processing completes, retrieve the logs for modules fnd.wf.bes% and fnd.sif% which will give logs only for the invocation through the framework. Also, you could retrieve the logs for the rest of the concurrent program processing using the Request ID.

To retrieve logs, log on to Oracle E-Business Suite as a user who has the System Administrator responsibility. Select **Oracle Applications Manager > Logs** from the Navigator to search your logs.

Troubleshooting SOAP Service Invocation Failure on Standalone JVM

When invoking a 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 more information, refer to the sample Java code described in *Invoking SOAP Services*, page 12-25.

Extending Seeded Java Rule Function for SOAP Services

Oracle E-Business Suite Integrated SOA Gateway allows developers to extend the seeded rule function `oracle.apps.fnd.wf.bes.WebServiceInvokerSubscription` for SOAP services using Java coding standards for more specialized processing.

Developers can extend the seeded rule function to override following methods:

- `preInvokeService`
- `postInvokeService`
- `addWSSecurityHeader`
- `setInputParts`
- `addCustomSOAPHeaders`

For detailed information about these methods, see *Oracle Workflow API Reference*.

Use the following steps to extend the seeded rule function:

1. Extend the methods using `oracle.apps.fnd.wf.bes.WebServiceInvokerSubscription`.
2. Upload the compiled custom class file at `$JAVA_TOP/oracle/apps/fnd/wf/bes/`.
3. Bounce the `oacore` and `oafm` servers.

4. Use the custom rule function `oracle.apps.fnd.wf.bes.xxxx` while creating the subscription.

Note that `xxxx` is the name of extended custom class. For example, `oracle.apps.fnd.wf.bes.CustomWebServiceInvoker`.

preInvokeService

This method is used for preprocessing before invoking a SOAP service.

```
protected String preInvokeService(Subscription eo,
                                   BusinessEvent event,
                                   WorkflowContext context)
throws BusinessEventException;
```

The 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 the business event `Generate` function.

This method can perform an 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 the input payload message is available.

postInvokeService

This method performs the post-processing after invoking a SOAP service.

```
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 an additional processing on the response and updates application state if required. The default implementation through the seeded Java rule function performs the following tasks:

- XSL transformation on a response or service output message based on `WFBES_OUT_XSL_FILENAME`
- Call back to Workflow Business Event System based on the `WFBES_CALLBACK_EVENT` and `WFBES_CALLBACK_AGENT` parameter values

addWSSecurityHeader

```
protected void addWSSecurityHeader(ArrayList headersList) throws
Exception;
```

This method adds WS-Security compliant header to the SOAP request. The default implementation through the Java seeded rule function adds the *UsernameToken* element

to the security header based on the event parameters `WFBES_SOAP_USERNAME`, `WFBES_SOAP_PASSWORD_MOD`, and `WFBES_SOAP_PASSWORD_KEY`, and sets the expiration time for the header in the `Timestamp` element based on the `WFBES_SOAP_EXPIRY_DURATION` parameter.

This method can be overridden to add any WS-Security header or have custom logic to retrieve user name and password to build the `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(String[] partNames, Hashtable<String,
Element> partValues) throws Exception;
```

Note: The method, `setInputParts(WSIFMessage, Input, String)` throws `Exception`; used in earlier releases is not supported in this release. Any subclass of `WebServiceInvokerSubscription` that implements this method should be modified to use the new method as explained here. Developers are required only to create the `org.w3c.dom.Element` objects for input part values and set it to the `Collections` object.

This `setInputParts` method can be optionally implemented in a subclass of `WebServiceInvokerSubscription` to set values for all input parts for the operation. The subclass then is used as the Java Rule Function for the "Invoke Web Service" event subscription.

This method gives the list of input part names for the operation that is invoked by that specific invocation instance as an array of `java.lang.String` in parameter `partNames`. Implementation of this method could set self-contained XML elements of type `org.w3c.dom.Element` to `partValues java.util.Hashtable` parameter for

each part name as key.

You can use a mix of event parameters, event payload, and extension of the `setInputParts` method to pass input part values for a service invocation. For example, one or more of the following combinations are possible:

- All input part values can be set from the implementation of the `setInputParts` method and event payload could be null.
- All input part values can be passed to the SOAP service invocation framework as business event parameters or payload. Typically the event payload will carry the largest part value like `<soap:Body>`.
- Combination of both above approaches.
- If some part values are passed as event parameters or payload and also set from the `setInputParts` method, the value from the `setInputParts` method prevails.

Input part values of type `org.w3c.dom.Element` for the SOAP service operation can be set in one of the following two ways:

- Pass the XML Element value as event parameters of name `WFBES_INPUT_<partname>` for each part.
- Method `setInputParts(String[], Hashtable (<String, Element>))` throws `Exception`; can be extended to generate and set XML elements corresponding to each Input part of the SOAP service operation that is invoked.

The following code snippet shows how this method is used to set the values for Input parts 'header' and 'body' by creating Element objects out of hand-coded XML element string:

```

final protected void setInputParts(String[] partNames, Hashtable<String,
Element>
partValues) throws Exception {
    String METHOD_NAME = CLASS_PREFIX+"setInputParts(String[],
Hashtable<String,Element>";
    writeLog(METHOD_NAME, "BEGIN", Log.PROCEDURE);
    String value = "<SOAHeader xmlns:ns1=\"http://xmlns.oracle.
com/apps/fnd/soapprovider/plsql/fnd_user_pkg/\>
<Responsibility>SYSTEM_ADMINISTRATOR<Responsibility>
<RespApplication>SYSADMINn<RespApplication>
<SecurityGroup><SecurityGroup>
<NLSLanguage><NLSLanguage>
    <Org_Id><Org_Id>
<SOAHeader>";
    partValues.put("header", getDocumentElement(value));
    value = "<InputParameters
xmlns=\"http://xmlns.oracle.
com/apps/fnd/soapprovider/plsql/fnd_user_pkg/testusername/\>
><X_USER_NAME>SYSADMIN</X_USER_NAME></InputParameters>";
    partValues.put("body", getDocumentElement(value));
    value = "<GetTheatersAndMovies
xmlns=\"http://www.example.com/whatsshowing\"
><zipCode>32822</zipCode><radius>10</radius></GetTheatersAndMovies>";
    partValues.put("parameters", getDocumentElement(value));
    writeLog(METHOD_NAME, "END", Log.PROCEDURE);
}

/**
 * This function is called to convert String to XML Element
 * @param data
 * @return
 * @throws Exception
 */

public Element getDocumentElement(String data)
throws Exception {
    String METHOD_NAME = "getDocumentElement(String)";
    Element ret = null;
    writeLog(CLASS_PREFIX + METHOD_NAME, "BEGIN", Log.PROCEDURE);
    DOMParser parser = new DOMParser();
    parser.parse(new StringReader(data));
    Document doc = parser.getDocument();
    if(doc != null) {
        ret = doc.getDocumentElement();
    }
    writeLog(CLASS_PREFIX + METHOD_NAME, "END", Log.PROCEDURE);
    return ret;
}

```

addCustomSOAPHeaders

```

protected void addCustomSOAPHeaders(ArrayList<Element> customHeaders)
throws Exception;

```

Note: The method addSOAPHeaders used in earlier releases is not supported in this release. Any subclass of WebServiceInvokerSubscription that implements this method should be modified to use the addCustomSOAPHeaders method as explained here. Developers are required only to create org.w3c.dom.Element objects for your custom SOAP header and set it to the

Collections object.

This `addCustomSOAPHeaders` method can be optionally implemented in a subclass of `WebServiceInvokerSubscription` to set custom SOAP headers for the SOAP request. The subclass then is used as the Java Rule Function for the "Invoke Web Service" event subscription.

Implementation of this method could set any number of self-contained XML elements of type `org.w3c.dom.Element` to customHeaders `java.util.ArrayList` parameter. All the XML elements will be added to the SOAP header.

This method helps to add a custom SOAP header to the SOAP request that is not defined in the input message for the WSDL operation that is invoked. For setting values to specific input parts as defined in WSDL operation's input message, use the `setInputParts` method.

The following code snippet shows how this method is used to set custom SOAP headers for a SOAP request:

```
final protected void addCustomSOAPHeaders(ArrayList customHeaders)
throws Exception {
    String custHdr = mEvent.getStringProperty("XXX_CUSTOM_HEADER");
    try {
        DocumentBuilderFactory factory = DocumentBuilderFactory.
newInstance();
        factory.setNamespaceAware(true);
        DocumentBuilder bldr = factory.newDocumentBuilder();
        Document doc = bldr.newDocument();
        doc = bldr.parse(new ByteArrayInputStream(custHdr.getBytes()));
        customHeaders.add((Element)doc.getFirstChild());
    }
    catch (Exception e) {
        throw e;
    }
}
```

Sample Codes

The following code shows how to extend the `addCustomSOAPHeaders` method to add any additional Header elements to a SOAP header that are not defined in WSDL:

```

final protected void addCustomSOAPHeaders(ArrayList customHeaders)
    throws Exception {
    String METHOD_NAME = CLASS_PREFIX+"addCustomSOAPHeaders
(ArrayList)";
    writeLog(CLASS_PREFIX + METHOD_NAME, "BEGIN", Log.PROCEDURE);

    // Add my own Custom header
    writeLog(METHOD_NAME, "Adding Custom header", Log.STATEMENT);
    addMyCustomHeader(customHeaders);

    // Add more headers if required to the ArrayList
    System.out.println("Adding Custom Headers in the sub-class");
    writeLog(METHOD_NAME, "END", Log.PROCEDURE);
}

private void addMyCustomHeader(ArrayList headersList)
    throws Exception {

    String METHOD_NAME = CLASS_PREFIX+"addMyCustomHeader
(ArrayList)";

    writeLog(METHOD_NAME, "BEGIN", Log.PROCEDURE);

    // Adding special SOAP Header to the request. This is required
only
    // if the WSDL's SOAP binding does not mandate the header but
it is
    // still required by the service. The XML element should be
self-sufficient
    // with all namespace declarations local to this element

    // In this case, the custom header is passed as an Event
Parameter and
    // set to the request to avoid hard-coding the header element
in code.
    // Any element can be passed at the time of raising the invoker
business event
    String custHdr = mEvent.getStringProperty("XXX_CUSTOM_HEADER");

    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());
            writeLog(CLASS_PREFIX + METHOD_NAME, "Successfully
added custom header 1", Log.STATEMENT);
        }
        catch (Exception e) {
            throw new BusinessException("Exception when
creating header element - "+e.getMessage());
        }
    }

    String custHdr2 = mEvent.getStringProperty
("XXX_CUSTOM_HEADER2");

    if (custHdr2 != null && !"".equals(custHdr2)) {
        try {

```

```

DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    factory.setNamespaceAware(true);
    DocumentBuilder bldr = factory.newDocumentBuilder();
    Document doc = bldr.newDocument();

    doc = bldr.parse(new ByteArrayInputStream(custHdr2.
getBytes()));

        // Add the element to the Headers list
        headersList.add((Element)doc.getFirstChild());
        writeLog(CLASS_PREFIX + METHOD_NAME, "Successfully
added custom header 2", Log.STATEMENT);
    }
    catch (Exception e) {
        throw new BusinessException("Exception when
creating header element - "+e.getMessage());
    }
}

writeLog(METHOD_NAME, "END", Log.PROCEDURE);
}

```

Other SOAP Service Invocation Usage Considerations

While implementing the framework to invoke SOAP services, some limitations need to be considered.

- WFBES_INPUT_<partname> parameter can only be passed at runtime during the event raise.
- The SOAP service invocation framework supports invoking only document-based services.
- Support One-to-One relationship of event subscriptions.

To successfully invoke SOAP 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 more implementation consideration about the framework, see Implementation Limitation and Consideration for the SOAP Service Invocation Framework, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

Using Service Invocation Framework to Invoke REST Services

REST Service Invocation Framework Overview

Similar to the SOAP service invocation framework, the REST service invocation framework also leverages Oracle Workflow Java Business Event System to provide an infrastructure for REST service invocation natively from Oracle E-Business Suite.

With this framework, an developer can directly interact with a REST service through the service endpoint, provided while defining the REST service metadata. The metadata will then be stored as subscription parameters and used later during the actual service invocation at runtime.

This framework supports REST service invocation through the GET or POST HTTP method with XML or JSON payload. It also supports HTTP Basic authentication security, error and exception handling, and test invocation features. Furthermore, it provides advanced configuration option allowing you to extend the HTTP header request with Signature and Digest header options, to add callback configuration, and to include additional REST service invoker parameters if required as part of the metadata for the REST service to be invoked.

In summary, the REST service invocation framework provides the following functionality:

- It relies on the Business Event System to create an event and event subscription and to parse a given REST service endpoint and resources store as subscription parameters.
- It supports REST service invocation through the GET or POST HTTP method with XML or JSON payload.
- It uses the Oracle Workflow seeded Java rule function `oracle.apps.fnd.wf.bes.RESTServiceInvokerSubscription` to help invoke REST services.

- It relies on the Oracle Workflow Test Business Event page to test service invocation by raising an invoker event from PL/SQL or Java and process synchronous and asynchronous subscriptions to the event.
- It utilizes the Error processing feature provided in the Business Event System to manage errors during subscription invocation and sends error notifications to the SYSADMIN user with the service definition, event, and error details.
- It utilizes the workflow notification system to send error notifications to and process responses from the SYSADMIN user.

For detailed information about Oracle Workflow, see the *Oracle Workflow User's Guide*, *Oracle Workflow Developer's Guide*, and the *Oracle Workflow Administrator's Guide*.

This section describes the information on how to invoke REST services using the REST service invocation framework. It includes the following topics:

Note: All outbound service invocation messages from Oracle E-Business Suite through service invocation framework are monitored using Service Invocation Monitor. See: *Monitoring and Managing Outbound Service Invocation Messages Using Service Invocation Monitor, Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

- Understanding REST Service Message Patterns, page 13-3
- Calling Back to Oracle E-Business Suite with REST Service Response, page 13-3
- Supporting REST Service Security, page 13-3
- Understanding REST Service Invocation Metadata, page 13-4
- Managing REST Service Invocation Errors, page 13-11
- Defining REST Service Invocation Metadata, page 13-11
- Invoking REST Services, page 13-20
- Testing REST Service Invocation, page 13-22
- An Example of Invoking a REST Service from a Java API, page 13-24
- Troubleshooting REST Service Invocation Failure, page 13-28
- Extending Seeded Java Rule Function for REST Services, page 13-29

Understanding REST Service Message Patterns

The REST service invocation framework supports Synchronous Request - Response message pattern and One - Way (Request Only) message pattern. These message patterns are supported in the same way as they are used in the SOAP service invocation framework.

For information on how these patterns work, see Understanding SOAP Service Message Patterns, page 12-3.

Calling Back to Oracle E-Business Suite with REST Service Response

The REST 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. This feature is implemented to support synchronous request - response service operation, if a REST service has an output or a response message.

When callback is required for your REST service invocation, you must define the following callback values:

- **Callback Agent** - An inbound agent to which the event with the REST service response message as the payload can be enqueued.
- **Callback Event** - A business event to be raised upon completion of the REST service with the service output message as the payload.

This callback mechanism takes the REST service invoker event's event key to enqueue the callback event to the callback agent you specified here for the response. If a workflow process invokes a REST service using a "Raise" event activity and waits for a REST service response using a "Receive" event activity, then the REST invoker event key should be the 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 the `WF_ENGINE.EVENT` API.

With callback event and agent, the REST service invocation can be integrated back with a waiting workflow process or any other module within Oracle E-Business Suite.

Supporting REST Service Security

When creating the event subscription for the REST service to be invoked, the developer needs to specify the user name and password information for the *HTTP Basic Authentication* REST service security so that the user credentials can be passed in the HTTP Header to authenticate the user before invoking the REST service.

Additionally, this REST service invocation framework supports the *Digest HTTP Header* and *Signature HTTP Header*. That is, you can optionally add another layer of

security to further define the invocation metadata by extending the HTTP header request with Signature and Digest header options during the event subscription creation.

Note that the REST service security is handled based on the *customization level*. This is similar to the existing behavior of the SOAP service invocation framework on handling the user name and password.

- If the Invoke REST Service event subscription's customization level is Core or Limit, and if the subscription owner has provided user name, then the Username field is read-only and cannot be updated. If the user name was not already supplied, the developer can update it if required.
- If the subscription's customization level is User, you can update the Username field. Password should always be editable regardless of the customization level.
- The Signature HTTP header related fields including PKCS #12 Keystore, Keystore Password, and Certificate Alias are always editable regardless of the customization level.

For more information about the REST service security, see Supporting REST Service Security and Configuring Security with Customization Level, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

Understanding REST Service Invocation Metadata

To be able to successfully invoke a REST service through the invocation framework, the developer needs to supply invocation metadata that will be used at runtime when the invocation actually happens. These metadata are provided in the event subscription creation for the triggering event to be occurred. This section explains the concepts and definitions of the invocation metadata in details.

When creating an event subscription in the Create Event Subscription page, the developer needs to first select a triggering event, enter related execution condition, and select "Invoke REST Service" as the action type before proceeding to the following pages to enter REST service invocation details:

Note: Oracle E-Business Suite Adapter with Oracle Integration has leveraged the REST service invocation framework for Business Event capabilities in integrations. For the event subscriptions created automatically from Oracle E-Business Suite Adapter with Oracle Integration, in the Create Event Subscription page you can only update the values of the Status and Phase fields. All other fields in the page are displayed as read-only fields and are not updatable.

For more information about the Oracle E-Business Suite Adapter with Oracle Integration, see *Using the Oracle E-Business Suite Adapter with*

Oracle Integration, available in the Oracle Cloud Library on the Oracle Help Center.

- **Create Event Subscription - Invoke REST Service Page**
This page provides the basic information of the REST service to be invoked. See: REST Service Details, page 13-5
- **Create Event Subscription - Invoke REST Service Advanced Configuration Page**
This page allows you to enter additional configuration information if needed for the REST service to be invoked. See: Advanced Configuration, page 13-9

REST Service Details

After specifying general subscription information in the Create Event Subscription page, you proceed to the Create Event Subscription - Invoke REST Service page where you must provide basic REST service information for the invocation.

Create Event Subscription - Invoke REST Service Page

Home Developer Studio **Business Events** Status Monitor Notifications Administration

Events Subscriptions Agents Systems

Business Events: Subscriptions > Create Event Subscription >

Create Event Subscription - Invoke REST Service Cancel Back Advanced Configuration Apply

Rest Service Details

* Resource Base
Resource Path

Service Endpoint

* HTTP Method POST
* Content Type XML
* Accept XML

Query Parameters

Enter Query Parameters and their values, if required.

Select Object: Delete | + | **

Name	Value
<input type="text"/>	<input type="text"/>

Table Diagnostics

REST Service Security

Enter Username and Password for HTTP Basic Authentication, if required.

Username
Password
Repeat Password

Rest Service Invoker

The Rule Function controls the behaviour of the subscription. Provide a Java Class name (<Package>. <Class>) for Java Rule Function and a PL/SQL stored procedure (<Package>. <Function>) for PL/SQL Rule Function.

Java Rule Function

Documentation

* Owner Tag
Owner Tag is same as the Application Short Name to which the Event or Subscription belongs. Example: FND, PO and so on.

* Owner Name

* Customization Level User

Description

Diagnostic Console

Cancel Back Advanced Configuration Apply

- **REST Service Details**

- Resource Base – Enter the base URI of the resource. It should start with either `http://` or `https://`.

The value entered here is stored in the subscription parameter:
`WFBES_REST_RESOURCE_BASE`

- Resource Path – Enter the relative URI path of the resource. By default, value `/` should be displayed.

The value entered here is stored in the subscription parameter:
`WFBES_REST_RESOURCE_PATH`

- Service Endpoint – The value of this field is populated automatically based on the resource base and resource path you entered earlier. This is the endpoint URL where the HTTP request would be sent at runtime.

For example,

- `WFBES_REST_RESOURCE_BASE = https://<host>:<port>/webservicess/rest/fnduserpkg`

Note that **fnduserpkg** is the service alias of a deployed REST service "User" (FND_USER_PKG).

- `WFBES_REST_RESOURCE_PATH = /testusername/`

At runtime, the service endpoint would be:

```
https://<host>:<port>/webservicess/rest/fnduserpkg
/testusername
```

- HTTP Method – Select either POST or GET from the drop-down list. By default, POST is selected.

The value entered here is stored in the subscription parameter:

`WFBES_REST_HTTP_VERB`

For example: `WFBES_REST_HTTP_VERB = POST`

- Content Type – Select either XML or JSON from the drop-down list. By default, XML is selected.

If XML is selected, then the value of Content-Type header is set to 'application/xml' in an HTTP request. Similarly, if JSON is selected, then the value of Content-Type header is set to 'application/json' in an HTTP request.

- Accept – Select either XML or JSON from the drop-down list. By default, XML is selected.

If XML is selected, then the value of Content-Type header is set to 'application/xml' in an HTTP request. Similarly, if JSON is selected, then the value of Content-Type header is set to 'application/json' in an HTTP request.

- **Query Parameters**

You can enter query parameter names and the associated values if required in the Query Parameters table as part of the REST service metadata. These parameters and their URL encoded values will be appended to the REST service endpoint. This provides the capability of adding new entries and deleting existing rows.

For example, at runtime these parameters and encoded values will be appended to a service endpoint as:

```
QueryParam1= URLEncode (QueryValue1)&QueryParam2= URLEncode
(QueryValue2)
```

The resultant query string is stored as the value for the subscription parameter `WFBES_REST_QUERY_STRING`. You can define query string for both GET and

POST HTTP Verbs.

- **REST Service Security for HTTP Basic Authentication**

Before invoking a REST service, in the REST Service Security region you need to specify the user name and password to be passed in the HTTP Header to authenticate the user.

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

Configuring Security Information with Customization Level

Oracle Workflow allows various levels of updates on business event and subscription based on the customization level. Similar to the existing behavior of the SOAP service invocation framework, the REST service security related fields are handled based on the customization level in the invoker's event subscription. Specifically, these security fields are:

- Username
- Password
- PKCS #12 Keystore
- Keystore Password
- Certificate Alias

Basically, if an invoker's event subscription with a customization level of Core or Limit, and if the user name is already supplied by the subscription owner, you cannot update this field. If the user name is not already supplied, you can update the user name if it's required. If the invoker's event subscription with a customization level of User, this Username field should be editable. The Password field in the REST Service Security region can always be updated regardless of the customization level.

For the PKCS #12 Keystore, Keystore Password, and Certificate Alias security fields in the Create Event Subscription - Invoke REST Service Advanced Configuration page, these fields are also editable regardless of the customization level.

For information about the customization level, see Supporting REST Service Security and Configuring Security with Customization Level, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

- **REST Service Invoker**

By default, the value of the seeded Java Rule Function for REST services is `oracle.apps.fnd.wf.bes.RESTServiceInvokerSubscription`.

Advanced Configuration

The REST service invocation framework allows you to optionally add more configuration parameters for the REST service to be invoked in the Create Event Subscription - Invoke REST Service Advanced Configuration page. These include parameters for HTTP headers, callback configuration, and the REST service invoker.

Create Event Subscription - Invoke REST Service Advanced Configuration Page

The screenshot shows the 'Create Event Subscription - Invoke REST Service Advanced Configuration' page. The breadcrumb trail is: Business Events: Subscriptions > Create Event Subscription > Create Event Subscription - Invoke REST Service > Create Event Subscription - Invoke REST Service Advanced Configuration. The page has three main sections: 1. HTTP Headers: Includes 'Select DigestAlgorithm to add Digest HTTP Header to the request, if required.' with a 'Digest Algorithm' dropdown and 'Provide following values to add Signature HTTP Header to the request, if required.' with a 'Signing Algorithm' dropdown. Below is a table with columns 'Name' and 'Value'. 2. Callback Configuration: Includes 'Callback Agent' and 'Callback Event' input fields with search icons. 3. REST Service Invoker Parameters: Includes a table with columns 'Name' and 'Value'. At the bottom, there is a 'Diagnostic Console' button and a footer with copyright information and links for 'About this Page' and 'Privacy Statement'.

- **HTTP Headers**

- **Digest Algorithm** – You can optionally select "SHA-512" or "SHA-256" from the drop-down list in this field. If a specific digest algorithm is chosen, then the "Digest" HTTP Header will be generated and added to the HTTP Header as part of the request message to be sent out.

For example, if "SHA-256" is selected as the algorithm, then this selected value is stored in the subscription parameter as follow:

- `WFBES_REST_HEADER_DIGEST = SHA-256`
- **Signing Algorithm** – Similar to the Digest Algorithm field, you can optionally select a signing algorithm value ("rsa-sha512", or "rsa-sha256") from the drop-down list. If a specific signing algorithm is chosen, then the "Signature" HTTP Header will be generated and added to the HTTP Header as part of the request message to be sent out.

For example, if "rsa-sha256" is selected as the algorithm, then this selected value is stored in the subscription parameter as follow:

- WFBES_REST_SIGN_ALGORITHM = rsa-sha256

Additionally, enter the following information for the Signature HTTP header:

HTTP Headers Region

HTTP Headers

Select DigestAlgorithm to add Digest HTTP Header to the request, if required.

Digest Algorithm

Provide following values to add Signature HTTP Header to the request, if required.

Signing Algorithm Certificate Alias

PKCS #12 Keystore No file chosen KeyID

Keystore Password Headers to be Signed

Repeat Password

- PKCS #12 Keystore – If the Signing Algorithm value is selected, you must locate and upload the keystore file in .p12 or .pfx format only.
- Keystore Password – Enter the keystore password.
- Certificate Alias – If the Signing Algorithm value is selected, you must enter the alias associated with the digital certificate used for signing the headers.
- KeyID – Provide the KeyID parameter for the Signature header. If no value is entered here, KeyID will be automatically generated.

The KeyID parameter value should be base64 encoded value of SHA-256 fingerprint of the associated digital certificate. The value is stored in the subscription parameter WFBES_REST_SIGN_KEYID.

- Headers to be Signed – Provide HTTP Headers that need to be signed for the Signature header. The default value is "content-type accept".

Note the value in this field is a lowercased list of HTTP Header fields, separated by a single space character. Any trailing spaces or multiple spaces between headers will be truncated. Any uppercase characters will be converted to lowercase characters after you click **Apply**.

- HTTP Headers – You can provide a list of HTTP Header name and its value to be added to an HTTP request.

Other subscription parameters defined for each HTTP Header is stored as WFBES_REST_HTTP_HEADER_<NAME>.

For example: WFBES_REST_HTTP_HEADER_Referer = https://www.example.com

The <NAME> value should not be Accept, Content-Type, Digest, Signature, and Authorization as they are handled by the REST service invocation framework. Similarly, Host, Date, and Created HTTP Headers are automatically added, you

should not include them for <NAME>.

To add new headers, click the + icon. Click **Delete** to remove an entry from the table.

- **Callback Configuration**

If a REST service has an output or a response message, you need to select an inbound workflow agent in the Callback Agent field, and the business event used for the callback in the Callback Event field.

- **REST Service Invoker Parameters**

The metadata entered here will be stored as subscription parameters and will be used later during the REST service invocation.

Managing REST Service Invocation Errors

The REST service invocation framework manages the invocation errors through the same error and exception handling mechanism used in the SOAP service invocation framework. That is, you must create an Error subscription with the following values to enable the error processing in the Business Event System:

- 'Error' source type
- 'Launch Workflow' action type
- 'WFERROR:DEFAULT_EVENT_ERROR2' workflow process

If there is an error or exception, HTTP Return Status Code should be captured and available to the consuming program. The SYSADMIN user will receive a notification about the REST service definition and error details. For more information on error handling during the service invocation, see *Managing REST Service Invocation Errors, Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

Defining REST Service Invocation Metadata

Similar to defining SOAP service invocation metadata, you need to log in to Oracle Workflow Business Event System to create an Invoker event and the Invoker event subscription for the REST service to be invoked at runtime. If it is required to have a response message, then you need to create a receive event and the receive event subscription.

Specifically, this section includes the following topics:

1. [Creating a REST Service Invoker Business Event, page 13-12](#)

Before creating an event subscription, you must first create an event. This Invoker

event serves as a request message (or a REST service input message) for a service that needs to be created.

2. Creating a Local Subscription to Invoke a REST Service, page 13-13

This event subscription indicates that when a triggering event occurs, the action item of this subscription is to invoke a REST service that will be defined as part of this subscription.

3. Creating an Error Subscription to Enable Error Processing for the REST Service, page 13-16

This error subscription enables error processing in the Business Event System that is used to communicate with the SYSADMIN user of an error condition in subscription processing.

4. Creating a Receive Event for REST Response (Optional), page 13-18

This step is required only if a REST service has an output or a response message to communicate or call back to Oracle E-Business Suite.

If a REST service does not require a response, neither a receive event nor a receive event subscription should be created.

5. Creating a Receive Event Subscription for REST Response (Optional), page 13-19

Once a receive event is in place, you must create an External subscription to the receive event to pass the REST service response message.

Step 1: Creating a REST Service Invoker Business Event

The first step of defining REST service invocation metadata is to create an invoker event that can be served as a request message (or a REST service input message) in a message pattern to send inquiries to a service.

Note: In this release, the service invocation framework supports the following types of service invocation:

- **One-way (request only)** - A consumer or client sends a message to a service, and the service does not need to reply.
- **Synchronous request-response** - This requires a response before an operation continues.

If an invoker event requires a response, then you must define a receive event to communicate or call back into Oracle E-Business Suite after a REST service is successfully invoked. See Supporting REST Service Security, page 13-3 and Supporting REST Service Security, page 13-3.

For more information about business events, see *Events, Oracle Workflow Developer's Guide*.

To create a REST service invoker event:

1. Log in to Oracle E-Business Suite as a user who has 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.xxx.user.restservice.invoke`
 - Display Name: Enter an event display name, such as `oracle.apps.xxx.user.RESTservice.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')
4. Click **Apply** to save your work.

Step 2: Creating a Local Subscription to Invoke the REST Service

Once an invoker event is created, you need to create a local subscription to the invoker event with **Invoke REST Service** action type. When a triggering event occurs, the Business Event System processes the subscription through the seeded Java function and invokes the REST service as indicated by the action type.

To create a local event subscription with 'Invoke REST Service' action type:

1. Log in to Oracle E-Business Suite as a user who has 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.xxx.user.RESTservice.invoke`
- Phase: 90

If the event is raised from Java, the phase number determines whether an event will be invoked right away or enqueued to the `WF_JAVA_DEFERRED` queue.

Note: If the invoker event is raised from PL/SQL, it is always deferred to the `WF_JAVA_DEFERRED` queue regardless of the phase because the subscription has a Java rule function that cannot be processed in the database.

- If the phase is greater than or equal to 100, then the event is enqueued to the `WF_JAVA_DEFERRED` queue and will be dispatched later.
- If the phase is less than 100, then the event is dispatched immediately to the Java Business Event System soon after a triggering event occurs.
- Status: Enabled
- Rule Data: Message
- Action Type: Invoke REST Service
- On Error: Stop and Rollback

4. Click **Next**. This opens the Create Event Subscription - Invoke REST Service page.

1. Enter the following REST service metadata information:

- Resource Base – Enter the base URI of the resource. For example, `http://<host>:<port>/webservices/rest/user`

Note that **user** is the service alias of a deployed PL/SQL REST service called "User" (`FND_USER_PKG`). This service contains a service operation called "Test User Name" (`testusername`).

- Resource Path – Enter the relative URI path of the resource. For example,

/testusername/

- Service Endpoint – The value of this field is populated automatically based on the resource base and resource path you entered earlier, such as `http://<host>:<port>/webservices/rest/user/testusername/`.
 - HTTP Method – Select either POST or GET from the drop-down list. By default, POST is selected.
 - Content Type – Select either XML or JSON from the drop-down list. By default, XML is selected.
 - Accept – Select either XML or JSON from the drop-down list. By default, XML is selected.
2. In the Query Parameters region, enter a desired parameter name and its value. Click '+' if required to add more rows.
 3. In the REST Service Security region, enter information in the Username and Password fields if appropriate.
 4. In the REST Service Invoker region, the default Java Rule Function name `oracle.apps.fnd.wf.bes.RESTServiceInvokerSubscription` is automatically populated.

Important: If you have extended the functionality of the seeded rule function, manually enter your custom function name here.

5. In the Documentation region, enter an application name or a program name that owns the subscription (such as 'Oracle Workflow') in the Owner Name field. Enter the program ID (such as 'FND') in the Owner Tag field.
6. If this is all the information required for your REST service invocation, click **Apply** to save your work.
7. If you want to add more configuration and callback information for your REST service invocation, click **Advanced Configuration** to display the Create Event Subscription - Invoke REST Service Advanced Configuration page.
8. In the Create Event Subscription - Invoke REST Service Advanced Configuration page, enter the following information in the HTTP Headers region:
 - Digest Algorithm – Select either "SHA-512" or "SHA-256" from the drop-down list.

- Signing Algorithm – Select either "rsa-sha512", or "rsa-sha256" from the drop-down list.

Additionally, enter appropriate values for the following fields corresponding to the Signing Algorithm value if it's selected:

- PKCS #12 Keystore – If the Signing Algorithm value is selected, click **Choose File** to browse and upload the keystore file in .p12 or .pfx format.
 - Keystore Password – Enter the keystore password twice.
 - Certificate Alias – If the Signing Algorithm value is selected, enter the alias associated with the digital certificate used for signing the headers.
 - KeyID – Provide the KeyID parameter for the Signature header.
 - Headers to be Signed – Provide HTTP Headers that need to be signed for the Signature header. The default value is "content-type accept".
 - HTTP Headers – You can provide a list of HTTP Header name and its value to be added to an HTTP request.
9. In the Callback Configuration region, enter the following fields:
 - Callback Agent - Select a desired inbound agent from the drop-down list.
 - Callback Event - Select a desired event from the drop-down list.
 10. In the REST Service Invoker Parameters region, enter desired parameter names and their associated values in the table.
 11. Click **Apply** to save your work.

For more information, see *Defining Event Subscriptions, Oracle Workflow Developer's Guide*.

Step 3: Creating an Error Subscription to Enable Error Processing for the REST Service

To create an error subscription, you must subscribe to the REST invoker event with the 'Launch Workflow' action type.

When a triggering event occurs, this subscription enables error processing in the Business Event System and communicates with the SYSADMIN user by sending a workflow notification with the REST service definition, event details, and error details.

Similar to the error processing used in the SOAP service invocation framework, this notification allows the SYSADMIN user to quickly identify and respond to the error.

Additionally, the SYSADMIN user can invoke the REST service again once the underlying issue is resolved, abort the errored event if needed, or reassign an errored notification to another user if appropriate.

To create an error subscription with 'Launch Workflow' action type:

1. Log in to Oracle E-Business Suite as a user who has 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.xxx.user.RESTservice.invoke`
 - Phase: This can be any phase number.
 - Status: Enabled

Note: While updating an event and an event subscription, for seeded events with a customization level of Limit, you can only update the status. For seeded product-specific events with a customization level of Core, you cannot update any properties. You can only view the subscription definition.

For information on how to use customization level, see Supporting REST Service Security, page 13-3.

- Rule Data: Key
 - Action Type: Launch Workflow
 - On Error: Stop and Rollback
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 a 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.
 7. Click **Apply**.

Step 4: Creating a Receive Event for REST Response (Optional)

Similar to the SOAP service invocation framework, if a REST service has an output or a response message to communicate or call back to Oracle E-Business Suite, and the Invoker event is raised from Java code with the subscription phase greater than or equal to 100 or if the event is raised from PL/SQL, then you should create a Receive event for callback to complete the invocation process. Additionally, create an external subscription to the Receive event to pass the REST service response.

Note: If it is raised from Java with subscription phase less than 100, then the REST service is invoked immediately and response is available to the calling program 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 a callback event. Hence, you may not need to create a receive event.

If a REST service does not require a response, then there is no need to create a receive event.

To create a receive event for a REST service:

1. Log in to Oracle E-Business Suite as a user who has the Workflow Administrator Web responsibility. Select the **Business Events** link, and choose **Events** in the horizontal navigation.
2. In the Events page, click **Create Event** to open another Create Event page.
3. Enter the following information in the Create Event page:
 - Name: Enter an event name, such as `oracle.apps.xxx.user.RESTservice.receive`
 - Display Name: Enter an event display name, such as `oracle.apps.xxx.user.RESTservice.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')
4. Click **Apply** to create a receive event.

Step 5: Creating a Receive Event Subscription for REST Response (Optional)

If a Receive event is created, you must create an external event subscription to pass the REST service response that is enqueued to the inbound workflow agent you selected for the callback configuration while defining the REST service metadata.

The subscription to the Receive event does not have to be with the "Launch Workflow" action type. It can be created with any action type if appropriate.

To create a receive event subscription for a REST service:

1. Log in to Oracle E-Business Suite as a user who has 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.xxx.user.RESTservice.receive`
 - Phase: any phase number
 - Status: Enabled
 - Rule Data: Key
 - Action Type: any action type
 - On Error: Stop and Rollback
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 any other action type is selected, then a different type of the create event subscription page is displayed. By entering an appropriate action type through the subscription page, you can launch a workflow process or invoke 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 a workflow type that is waiting for the response
 - Workflow Process: Enter a workflow process that is waiting for the response
 - Priority: Normal
6. In the Documentation region, enter an application or a program name in the Owner Name field (such as 'Oracle Workflow'). Enter an application or a program ID in the Owner Tag field (such as 'FND').
7. Click **Apply**.

Invoking REST Services

REST service invocation from Oracle E-Business Suite can be from a PL/SQL or Java layer:

Service Invocation from PL/SQL

1. An 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 the *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.RESTServiceInvokerSubscription`.
3. The Business Event System enqueues the event message to the `WF_JAVA_DEFERRED` queue. The Java Deferred Agent Listener then dequeues and processes the subscription whose Java rule function invokes the REST service.
4. If callback event and agent parameters are mentioned, the REST service response is communicated back to Oracle E-Business Suite using the callback information. The

Java Deferred Agent Listener process that runs on the Concurrent Manager (CM) tier invokes the REST service.

Service Invocation from Java

1. A Java application raises a business event using Java method `oracle.apps.fnd.wf.bes.BusinessEvent.raise` either from an OA Framework page controller/AMImpl or Java code running on the Concurrent Manager tier.
2. Since the event is raised from Java where the subscription's seeded Java Rule Function `oracle.apps.fnd.wf.bes.RESTServiceInvokerSubscription` is accessible, whether the rule function is processed inline or deferred is determined by the phase of the subscription.
 - If the invoker subscription is created with the Phase value greater than or equal to 100, the event is enqueued to the `WF_JAVA_DEFERRED` queue.
 - If the invoker subscription is created with the Phase value less than 100, the event is dispatched inline.

If the event is raised from an OA Framework page, the dispatch logic runs within `OACORE WebLogic Server`.

Note: If the REST service invoker event is raised from Java on the application tier, and the invoker subscription is synchronous with the subscription phase value less than 100, then the REST service is invoked as soon as the event is raised. If the invocation is successful, the response can be read by the calling application and is available immediately by using method `BusinessEvent.getResponseData()`.

`oracle.apps.fnd.wf.bes.BusinessEvent.raise` throws `oracle.apps.fnd.wf.bes.BusinessEventException` if there are any issues while invoking a REST service inline. `BusinessEventException` object internally stores the underlying root cause exception within a `linkedException` object. In order to see the complete exception details, print the exception stack trace from `BusinessEventException.getLinkedException()`.

If the event is raised from Java with the subscription phase value greater than or equal to 100 or if the event is raised from PL/SQL, the event message will be enqueued to the `WF_JAVA_DEFERRED` queue. If the REST service has an output or a response message, a callback event with callback agent is required to receive the output message into Oracle E-Business Suite.

Testing REST Service Invocation

The Oracle Workflow Test Business Event page used for testing SOAP service invocation is also used for testing REST service invocation by raising a test event from either a Java or PL/SQL layer and by processing synchronous or asynchronous subscriptions to that event. You can easily validate whether a REST service can be successfully invoked from the concurrent manager tier and OACORE WebLogic Server.

Use the Test Business Event page to test an event by raising it from a PL/SQL API or Java.

- For an invoker event raised using the **Raise in Java** option, the REST service is invoked from the OACORE WebLogic server if the subscription phase is less than 100.

If the REST service is successfully invoked, the Test Business Event page reloads and displays the XML/JSON Response region right after the XML/JSON Content field.

If there is a runtime exception when invoking the REST service using synchronous subscription, the exception message is shown on the Test Business Event page.

- For an invoker event raised using the **Raise in PLSQL** option, the REST service is invoked from the concurrent manager tier. The raised event will be enqueued to WF_JAVA_DEFERRED and then dispatched by the Workflow Java Deferred Agent Listener.

The seeded Java rule function uses the callback event and agent to communicate the response or REST service output message back to Oracle E-Business Suite through Business Event System.

Note: Since the Workflow Java Deferred Agent Listener is responsible for dispatching the subscription and invoking REST services from the concurrent manager tier, ensure that the 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 the Workflow Java Deferred Agent Listener to view its status.

Testing REST Service Invocations

To access the Test Business Event page, log in to Oracle E-Business Suite as a user who has the Workflow Administrator Web responsibility, and select **Business Events** from the navigation menu. After a search to locate the event that you want to raise for testing the REST service invocation, click the **Test** icon. This displays the Test Business Event page where you can raise the event with a 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 the XML/JSON response message if appropriate.

To test an event invocation:

1. Log in to Oracle E-Business Suite as a user who has the Workflow Administrator Web responsibility and select **Business Events**.
2. Search on a business event that you want to run the test, such as `oracle.apps.xxx.user.RESTservice.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/JSON
 - XML/JSON Content: Enter appropriate XML or JSON information as an input message. For example, you can enter the following content in JSON format:

```
{
  "TESTUSERNAME_Input": {
    "RESTHeader": {
      "Responsibility": "SYSTEM_ADMINISTRATOR",
      "RespApplication": "SYSADMIN",
      "SecurityGroup": "STANDARD",
      "NLSLanguage": "AMERICAN",
      "Org_Id": "202"
    },
    "InputParameters": {
      "X_USER_NAME": "operations"
    }
  }
}
```

7. Click **Raise in Java** to raise an event from the OACORE WebLogic Server.
If the REST service is successfully invoked, a confirmation message appears on top of the Test Business Event page indicating that the event (`oracle.apps.xxx.user.RESTservice.invoke`) has been successfully raised.
This test page reloads and displays the XML/JSON Response region right after the XML/JSON Content field.
8. Click **Raise in PLSQL** to raise an event from the concurrent manager tier.
9. If errors occur, the Event Error Details region appears letting you view the error details.

Viewing Error Details in the Test Business Event Page

If any errors occur while testing the service invocation, an error message appears indicating that errors occurred while dispatching the event and detailed information is shown in the Event Error Details region.

The Event Error Details region lets you quickly view error information through the same page for easier debugging.

Optionally, to see detailed log messages that capture each occurrence in sequential order for service invocation, before testing the invocation, you can enable the diagnostics and logging feature to directly display on-screen logs in the test page. For instructions about how to turn on this logging feature, see *Troubleshooting SOAP Service Invocation Failure on OACORE WebLogic Server*, page 12-36.

For more information about testing business events, see *To Raise a Test Event*, *Oracle Workflow Developer's Guide*.

An Example of Invoking a REST Service from a Java API

Invocation Scenario Using the REST Service Invocation Framework

Take a PL/SQL API called "User" (FND_USER_PKG) that contains a service operation "Test User Name" (testusername) as an example to explain a REST service invocation through the REST service invocation framework.

In this example, this "User" API is deployed as a REST service with alias "user". We will invoke the deployed testusername service operation from a Java class using the REST service invocation framework. The invoker business event will be raised from this Java class. Based on the FND_USER_PKG.TESTUSERNAME REST service metadata defined in the invoker event subscription, it will be invoked. The corresponding response message is also received at the same session of the invocation.

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

Steps to Invoke REST Service

Based on the invocation scenario, the invocation through the REST service invocation framework includes the following steps:

1. Create an invoker business event to invoke a REST service.
See: *Creating an Invoker Business Event*, page 13-25.
2. Create an invoker business event subscription to invoke a REST service.
See: *Creating an Invoker Business Event Subscription*, page 13-25.
3. Create a Java class to raise the invoker business event.
See: *Creating a Java Class to Raise the Invoker Business Event*, page 13-26.

4. Compile and run the Java class to obtain the response.
See: Compiling and Running the Java Class, page 13-28.

Creating an Invoker Business Event

To invoke a deployed REST service `FND_USER_PKG` using the REST service invocation framework, the first step is to create an invoker event. This invoker event serves as a request message for the `FND_USER_PKG` REST service to be created during the event subscription.

For detailed step-by-step instructions on creating an invoker event, see *Creating a REST Service Invoker Business Event*, page 13-12.

Creating an Invoker Business Event Subscription

This event subscription indicates that when a triggering event from a Java class occurs, the action item of this subscription is to invoke the deployed `FND_USER_PKG` REST service that is defined in this step as part of the subscription.

Perform the following steps to create an invoker event subscription:

1. Enter event subscription information with the following key values in the Create Event Subscription page for this example:
 - Action Type: Invoke REST Service
 - Phase: 10
2. In the Create Event Subscription - Invoke REST Service page, enter the following service metadata information in the REST Service Details region:
 - Resource Base – Enter the base URI of the resource, such as `http://<host>:<port>/webservices/rest/user` in this example.
Note that **user** is the service alias of the deployed `FND_USER_PKG` REST service.
 - Resource Path – Enter the relative URI path of the resource. For example, `/testusername/`
Note that `testusername` is the service operation contained in the deployed `FND_USER_PKG` REST service.
 - Service Endpoint – The value of this field is populated automatically based on the resource base and resource path you entered earlier, such as `http://<host>:<port>/webservices/rest/user/testusername/`.
 - HTTP Method – Leave the default POST unchanged.

- Content Type – Leave the default XML unchanged.
 - Accept – Leave the XML unchanged.
3. Leave the Query Parameters region blank, and enter security information in the REST Service Security region.
 4. In the Rest Service Invoker region, leave the default Java Rule Function value unchanged, and enter appropriate values in the Documentation region.

For more information on creating an invoker event subscription, see [Creating a Local Subscription to Invoke a REST Service](#), page 13-13.

Creating a Java Class to Raise the Invoker Business Event

The following sample Java code raises a business event that invokes a REST 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 InvokeREESTService {
    static Log mLog;
    static WorkflowContext mCtx;

    public InvokeREESTService() { }

    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 {
            // Proxy host and port, if required
            //System.setProperty("http.proxyHost", args[1]);
            //System.setProperty("http.proxyPort", args[2]);

            //Replace business event and event key
            event = new BusinessEvent (<business event>,<event key>);

            // Input XML or JSON message for REST Web Service
            String input = null;
            input = "{ \"TESTUSERNAME_Input\": { \"RESTHeader\": {
\\\"Responsibility\\\": \"SYSTEM_ADMINISTRATOR\", \"RespApplication\": {
\\\"SYSADMIN\\\", \"SecurityGroup\": \"STANDARD\", \"
NLSLanguage\": \"AMERICAN\", \"Org_Id\": \"202\" }, \"
InputParameters\": { \"X_USER_NAME\": \"operations\" } } } }";

            event.setData(input);

            event.raise(conn);
            conn.commit();

            Object resp = event.getResponseData();
            if (resp != null) {
                String respStr = resp.toString();
                // Process REST web service response here
                // Response could be XML or JSON message
            }
            else {
                // Either REST web service invocation failed and no
                exception was thrown
                // or the web service is one-way or asynchronous and did
                not return
                // a valid response
                System.out.println("No response received");
            }
        }
    }
}

```

```

    }
    }
    catch (BusinessEventException e) {
        // Use appropriate logging mechanism as per your coding
standards
        // instead of System.out.println
        System.out.println("Exception occurred " + e.
getLinkedException().getMessage());
        // Print the complete exception stack to log file for
troubleshooting
        // Most importantly, if an exception occurred, do not
proceed to process the
        // response
        e.getLinkedException().printStackTrace();
    } catch (Exception e) {
        // Use appropriate logging mechanism as per your coding
standards
        // instead of System.out.println
        System.out.println("Exception occurred " + e.getMessage());
        // Print the complete exception stack to log file for
troubleshooting
        // Most importantly, if an exception occurred, do not
proceed to process the
        // response
        e.printStackTrace();
    }
}
}
}

```

Important: When invoking a REST service using the REST service invocation framework, the invoker business event is raised using the `oracle.apps.fnd.wf.bes.BusinessEvent.raise (Connection)` method that requires a JDBC connection to be passed.

To get the JDBC connection, always use the current application context object available for your scenario. For example, if a REST service invocation is from an OA Framework page, then get the JDBC connection from the `OAPageContext` object. If it is from a concurrent program, get the JDBC connection from the `CpContext` object. You should not create a `WorkflowContext` in these situations. Otherwise, a duplicate application context will be unnecessarily created. A new `WorkflowContext` should be created only if JDBC connection is not already available through other means.

Compiling and Running the Java Class

In this step, you need to compile and run the process to obtain the response of the REST service invocation using standard `javac` and `java` commands from the command line.

Troubleshooting REST Service Invocation Failure

Similar to the SOAP service invocations, REST services can also be invoked through one of the following tiers:

- OACORE WebLogic Server
- Concurrent Manager (CM) Tier JVM
- Standalone JVM

As the REST service to be invoked resides outside the firewall and the running host does not have direct access to the REST service endpoint to send the request, to successfully invoke the REST service from Oracle E-Business Suite, you need to set up and configure the proxy parameters for each tier that the service invocation may occur. See: Setup Tasks, *Oracle E-Business Suite Integrated SOA Gateway Implementation Guide*.

At runtime, if a REST service invocation fails, an exception is thrown and the invoker event is enqueued to the WF_ERROR queue. Since the service can be invoked from any of the three tiers, the troubleshooting is based on the three tiers where the REST service invocation may occur. For troubleshooting details, refer to the troubleshooting information described earlier for the SOAP service invocation framework at Troubleshooting SOAP Service Invocation Failure, page 12-35.

Note that when troubleshooting the invocation failure from OACORE WebLogic Server, you can review the on-screen log message if the on-screen logging feature is enabled.

For example, the following log indicates that the service invocation is completed with callback response message enqueued to the WF_WS_JMS_IN inbound queue if the callback event is set to "receive event" and the callback agent 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, corr
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 information on enabling the on-screen logging feature and viewing the on-screen logs, see: Troubleshooting SOAP Service Invocation Failure on OACORE WebLogic Server, page 12-36.

Extending Seeded Java Rule Function for REST Services

The REST service invocation framework allows you to extend the seeded rule function `oracle.apps.fnd.wf.bes.RESTServiceInvokerSubscription` for REST services using Java coding standards for more specialized processing.

Specifically, you can extend the seeded rule function to override the following methods:

- `preInvokeService`

- `postInvokeService`

For detailed information about these methods, see *Oracle Workflow API Reference*.

Use the following steps to extend the seeded rule function:

1. Extend the methods using `oracle.apps.fnd.wf.bes.RESTServiceInvokerSubscription`.
2. Upload the compiled custom class file at `$JAVA_TOP/oracle/apps/fnd/wf/bes/`.
3. Bounce the `oacore` and `oafm` servers.
4. Use the custom rule function `oracle.apps.fnd.wf.bes.xxxx` while creating the subscription.

Note that `xxxx` is the name of extended custom class. For example, `oracle.apps.fnd.wf.bes.CustomWebServiceInvoker`.

preInvokeService

This method is used for preprocessing before invoking a REST service.

```
protected String preInvokeService(Subscription eo,
                                   BusinessEvent event,
                                   WorkflowContext context)
throws BusinessEventException;
```

The 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 the business event `Generate` function.

postInvokeService

This method performs the post-processing after invoking a REST service.

```
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 an additional processing on the response and updates application state if required. The default implementation through the REST service seeded Java rule function performs the callback feature to the Workflow Business Event System based on the values provided for the Callback Event and Callback Agent.

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 are discussed in this chapter:

- Java Annotations, page A-4
- PL/SQL Annotations, page A-17
- 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-115
- Glossary of Annotations, page A-120

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-120
- rep:scope, page A-123
- rep:product, page A-124
- rep:implementation, page A-125

Only required for Java business service objects; not required for plain Java or SDOs.

- rep:displayname, page A-125
- rep:service, page A-142
- rep:servicedoc, page A-143

Optional Class-level Annotations

- link, page A-129
- see, page A-130
- rep:lifecycle, page A-127
- rep:ihelp, page A-131
- rep:category, page A-132

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. For more information, 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-128
- rep:standard, page A-135
- rep:metalink, page A-132
- rep:synchronicity, page A-144

Required Method-level Annotations

- must begin with description sentence(s), page A-120
- param, page A-138
Use only when applicable and when other tags such as @see and @rep:metalink do not provide parameter explanations.
- return, page A-139 (if applicable)
- rep:paraminfo, page A-139
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:displayname, page A-125
- rep:businessevent, page A-141 (if an event is raised)

Optional Method-level Annotations

- link, page A-129
- see, page A-130
- rep:scope, page A-123
- rep:lifecycle, page A-127
- rep:compatibility, page A-128
- rep:category, page A-132
Use BUSINESS_ENTITY at the method level only when a class methods have heterogeneous business entities.

- `rep:ihelp`, page A-131
- `rep:metalink`, page A-132
- `rep:appscontext`, page A-145
- `rep:primaryinstance`, page A-146
- `rep:httpverb`, page A-136

Use this annotation to specify the HTTP Verbs suitable for a Java method. See: Annotations for Java Bean Services, page A-6 and Annotations for Application Module Services, page A-6.

- `rep:synchronicity`, page A-144

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-132.
- `@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-136.
- `@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-139.

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-132.
- `@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-136.
- `@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-139.

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 OAAApplicationModuleImpl {
...
}
```

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:businessesevent 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} {@rep:required}
     *
     * @rep:scope public
     * @rep:displayname Receive Purchase Order Items
     * @rep:businessesevent 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:

```

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 specification 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

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-120
- `rep:scope`, page A-123
- `rep:product`, page A-124
- `rep:displayname`, page A-125
- `rep:category`, page A-132

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-141 (if an event is raised)

Optional Class-level Annotations

- `link`, page A-129
- `see`, page A-130
- `rep:lifecycle`, page A-127
- `rep:compatibility`, page A-128
- `rep:ihelp`, page A-131
- `rep:metalink`, page A-132

Required Method-level Annotations

- must begin with description sentence(s), page A-120
- `param`, page A-138
Use only when applicable and when other tags such as `@see` and `@rep:metalink` do not provide parameter explanations.
- `return`, page A-139 (if applicable)
- `rep:displayname`, page A-125
- `rep:paraminfo`, page A-139

- `rep:businesssevent`, page A-141 (if an event is raised)

Optional Method-level Annotations

- `link`, page A-129
- `see`, page A-130
- `rep:scope`, page A-123
- `rep:lifecycle`, page A-127
- `rep:compatibility`, page A-128
- `rep:category`, page A-132

Use `BUSINESS_ENTITY` at the method level only when a class methods have heterogeneous business entities.

- `rep:ihelp`, page A-131
- `rep:metalink`, page A-132
- `rep:appscontext`, page A-145
- `rep:primaryinstance`, page A-146

Template

You can use the following template when annotating PL/SQL files:

Note: Annotation for PL/SQL APIs can start with either `/**` or `/*#`.

```

.
.
.
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 on 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 and committing, you will need to use FNDLOAD to recreate 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-120

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-123
- rep:product, page A-124
- rep:displayname, page A-125

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-132
- rep:businesssevent, page A-141 (if an event is raised)

Note: There is no required method-level annotations for concurrent programs.

Optional Class-level Annotations

- link, page A-129
- see, page A-130
- rep:lifecycle, page A-127
- rep:compatibility, page A-128
- rep:ihelp, page A-131
- rep:metalink, page A-132
- rep:usestable, page A-134
- rep:usesmap, page A-146

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>
 */
```

Note: Annotation for concurrent programs can start with either `/**` or `/*#`.

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
 */

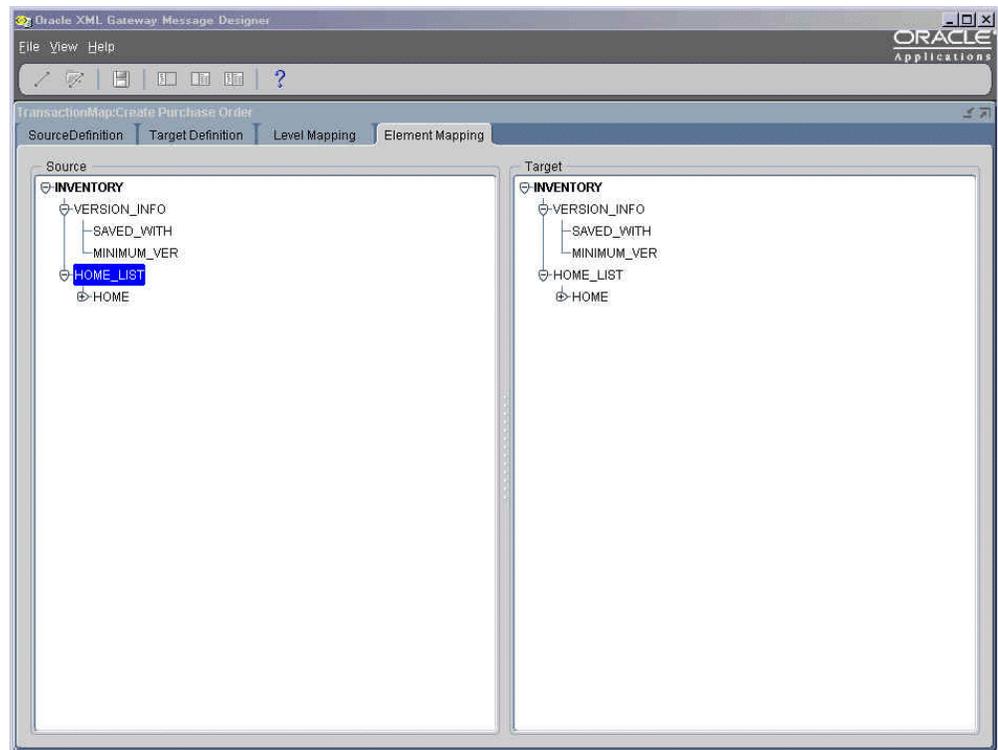
```

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.

Oracle XML Gateway Message Designer Form with Element Mapping Tab

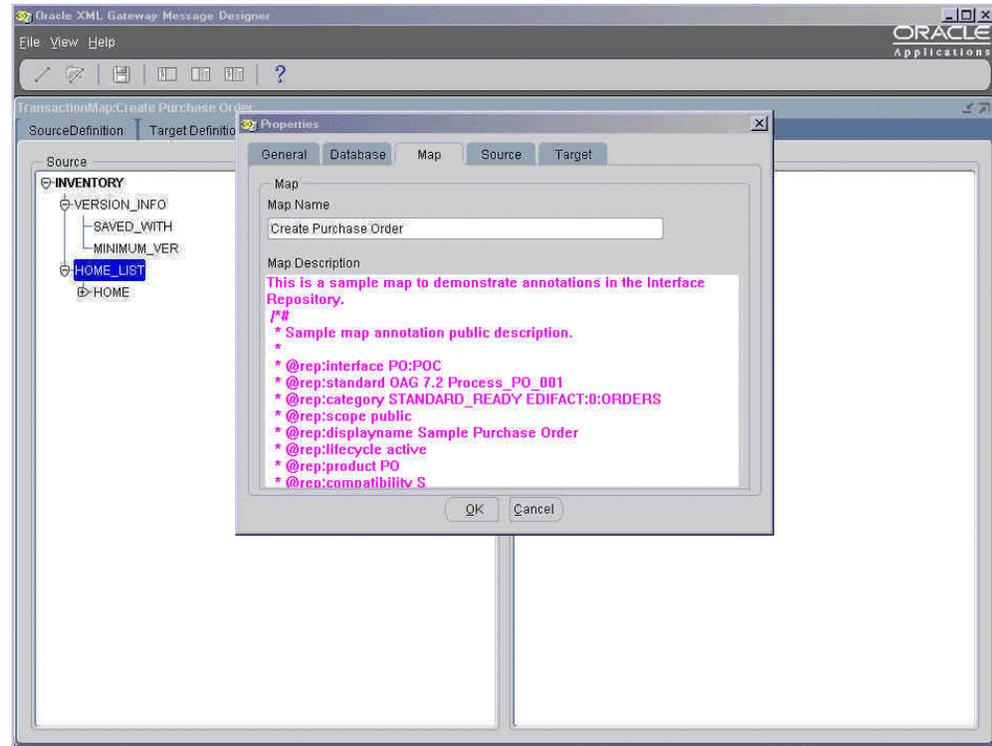


2. Find out which Internal Transaction Type, Subtype, Standard, and Direction this particular map is associated with. Note that this entry must exist in XML Gateway to be loaded into the Integration Repository.

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 with 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-132 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.

- 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-120
- rep:scope, page A-123
- rep:product, page A-124
- rep:displayname, page A-125

- rep:category, page A-132
- rep:standard, page A-135
- rep:interface, page A-137
- rep:businesssevent, page A-141 (if an event is raised)
- rep:direction, page A-141

Note: There is no required method-level annotations for XML Gateway.

Optional Class-level Annotations

- link, page A-129
- see, page A-130
- param, page A-138

Use only when applicable and when other tags such as @see and @rep:metalink do not provide parameter explanations.

- rep:paraminfo, page A-139
- rep:lifecycle, page A-127
- rep:compatibility, page A-128
- rep:ihelp, page A-131
- rep:metalink, page A-132
- rep:synchronicity, page A-144

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-125
- [rep:scope](#), page A-123
- [rep:product](#), page A-124
- [rep:category BUSINESS_ENTITY](#), page A-132

Optional Annotations

- [link](#), page A-129
- [see](#), page A-130
- [rep:lifecycle](#), page A-127
- [rep:compatibility](#), page A-128
- [rep:ihelp](#), page A-131
- [rep:metalink](#), page A-132

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-132.

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	Display Name	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	Display Name	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	Display Name	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	Display Name	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	Display Name	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	Display Name	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	Display Name	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	Display Name	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	Display Name	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 processing, 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	Display Name	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	Display Name	Description
DPP_TRANSACTION_APPR OVAL	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_AS SOCIATION	Maintenance Asset Activity Association	Maintenance Asset Activity Association
EAM_ASSET_ACTIVITY_SU PPRESSION	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_G ROUPS	Maintenance Asset Attribute Group	Maintenance Asset Attribute Group
EAM_ASSET_ATTRIBUTE_V ALUE	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_OPE RATION	Maintenance Work Completion	Maintenance Work Completion
EAM_DEPARTMENT_APPR OVER	Maintenance Department Approver	Maintenance Department Approver - User or responsibility

Business Entity Code	Display Name	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	Display Name	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_EDITION_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_ATTRIBUTES_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	Display Name	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	Display Name	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	Display Name	Description
FEM_XDIM_FINANCIAL_ELEMENT	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	Display Name	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	Display Name	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	Display Name	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 at runtime.
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	Display Name	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	Display Name	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_COMBINATIONS	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	Display Name	Description
GMD_FORMULA	Process Manufacturing Formula	Process Manufacturing Formula
GMD_OPERATION	Process Manufacturing Operation	Process Manufacturing Operation
GMD_OUTBOUND_APIS_P UB	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_R ULE	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	Display Name	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_CONVERSION	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	Display Name	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	Display Name	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	Display Name	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	Display Name	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	Display Name	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_FUNDCAPTURE_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	Display Name	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	Display Name	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	Display Name	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	Display Name	Description
INV_COUNT	Material Count	Material Count
INV_IC_TRANSACTION_FLOW	Inventory Intercompany Invoicing Transaction Flow	It is an implementation 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	Display Name	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_INVENTORY	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_PREFERENCES	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	Display Name	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	Display Name	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	Display Name	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	Display Name	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	Display Name	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).
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 invocation of the executable object.
LSH_EXECUTION_FWK	Life Sciences Data Hub Execution Job	An entity that provides the surround and the set of rules for the invocation of an object. Examples are Job Submission API, Job Log API, etc

Business Entity Code	Display Name	Description
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 invocation 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.
LSH_PARAMETERSSET	Parameter Set	A collection of interrelated Parameters

Business Entity Code	Display Name	Description
LSH_PLANNEDOUTPUT	Life Science Data Hub Planned Output	A Planned Output is an expected output when an LSH object is processed. 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 processed 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.
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.

Business Entity Code	Display Name	Description
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.
MSC_PLANNING_SUPPLY_DEMAND	Collaborative Planning Supply / Demand	This interface is used to create any supply / demand records in Collaborative Planning.

Business Entity Code	Display Name	Description
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	Display Name	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	Display Name	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	Display Name	Description
OKL_INVESTMENT_PROGR AM	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_PROGR AM	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_QUOT E	Lease Termination Quote	Lease Termination Quote
OKL_THIRD_PARTY_BILLI NG	Lease Third Party Billing	Lease Third Party Billing
OKL_UNDERWRITING	Manage Underwriting	It manages credit applications and lines.
OKL_VENDOR_RELATIONS HIP	Manage Vendor Relationship	It manages vendor accounts and agreements.

Business Entity Code	Display Name	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_FULFILLMENTS	Subscription Fulfillment Schedule	Sales Order Fulfillment Schedules will be defined for the subscription contracts that is for scheduling the Creation of Sales Orders and its fulfillment.
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.
OKS_SALES_CREDITS	Service Contracts Sales Credit	Quota/Non-Quota Sales credits percentage will be defined for the Salesperson(s) who are responsible for the service, subscription, warranty, and extended warranty contract.
OKS_TERMINATE	Service Contracts Termination	Terminating a subline, a line, or an entire contract against the customer with the Termination reason and Current or Future Termination date.

Business Entity Code	Display Name	Description
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
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.

Business Entity Code	Display Name	Description
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.
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_USAGE	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

Business Entity Code	Display Name	Description
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_OFFERING_RESOURCES_CHKLIST	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
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

Business Entity Code	Display Name	Description
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,
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.

Business Entity Code	Display Name	Description
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
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

Business Entity Code	Display Name	Description
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
PAY_PAYROLL_TABLE_RECORDABLE_EVENT	Payroll Table Recordable Event	Payroll Table Recordable Event
PAY_PERSONAL_PAYMENT_METHOD	Personal Payment Method	Personal Payment Method
PAY_PROVINCIAL_MEDICAL_ACCOUNT	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

Business Entity Code	Display Name	Description
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
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

Business Entity Code	Display Name	Description
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_STRUCTURE	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
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

Business Entity Code	Display Name	Description
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
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

Business Entity Code	Display Name	Description
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
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

Business Entity Code	Display Name	Description
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.
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

Business Entity Code	Display Name	Description
PER_RECRUITMENT_ACTIV ITY	Recruitment Activity	Recruitment Activity
PER_SALARY_PROPOSAL	Salary Proposal	Salary Proposal
PER_SALARY_SURVEY	Salary Survey	Salary Survey
PER_SCORECARD_SHARIN G	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_RO LE	HR Supplementary Role	HR Supplementary Role
PER_VACANCY	Vacancy	Vacancy
PER_VACANCY_REQUISITI ON	Vacancy Requisition	Vacancy Requisition
PER_WORK_COUNCIL_ELE CTION	Work Council Election	Work Council Election
PER_WORK_INCIDENT	Work Incident	Work Incident
PN_CUSTOMER_SPACE_AS SIGNMENT	Customer Space Assignment	Customer Space Assignment
PN_EMPLOYEE_SPACE_ASS GNMENT	Employee Space Assignment	Employee Space Assignment
PN_INDEX_LEASES	Index Rent	Index Rent Agreement: It manages creation and updates of index rents for Oracle Projects.

Business Entity Code	Display Name	Description
PN_LEASE	Lease	Lease Agreement: This describes the creation and updates of lease and payment terms for Oracle Projects.
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_VARIABLE_RENTS	Variable Rent	Variable Rent Agreement: It defines creation and updates of rent, overrides variable rent calculations, create breakpoints, constraints, allowances, and abatements as well as generate period for variable rent.
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_NOTIFICATION	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

Business Entity Code	Display Name	Description
PO_ATTACHMENTS	Procurement Attachments	Procurement Attachments
PO_AUCTION	Auction	Auction
PO_AWARD	Award	Award
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

Business Entity Code	Display Name	Description
PO_CONTRACT_PURCHASE_AGREEMENT	Contract Purchase Agreement	Contract Purchase Agreement
PO_CONTRACT_TEMPLATE	Purchasing Contract Template	Purchasing Contract Template
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

Business Entity Code	Display Name	Description
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
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

Business Entity Code	Display Name	Description
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
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_TRANSACTION_TEMPLATE	Position Control Transaction Template	Position Control Transaction Template

Business Entity Code	Display Name	Description
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
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.

Business Entity Code	Display Name	Description
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
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

Business Entity Code	Display Name	Description
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
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.

Business Entity Code	Display Name	Description
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
WIP_COMPLETION_TRANS ACTION	WIP Assembly Completion	Business Entity for Assembly Completion in WIP
WIP_EMPLOYEE_LABOR_R ATE	WIP Employee Labor Rate	WIP Employee Labor Rate
WIP_MATERIAL_TRANSAC TION	WIP Material Transaction	Business Entity for Material Transaction in WIP
WIP_MOVE_TRANSACTIO N	WIP Shopfloor Move	WIP Shopfloor Move
WIP_PARAMETER	Work in Process Setup	Work in Process Setup
WIP_PRODUCTION_LINE	Production Line	Production Line

Business Entity Code	Display Name	Description
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_BULK_PACK	Warehouse Bulk Pack Custom API	<p>This object can be used to specify custom method to decide:</p> <ul style="list-style-type: none"> • Whether multiple bulk tasks can be loaded to the same LPN or not • Whether for a PJM enabled organization the LPNs across projects and tasks can be consolidated or not
WMS_CONTAINER	Warehouse Management License Plate	Warehouse Container and License Plate Management
WMS_DEPLOY	Warehouse Management System Deployment Check	Object to get the WMS deployment mode and related standalone and LPN installation utilities.
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

Business Entity Code	Display Name	Description
WMS_DOCK_APPOINTMENTS	WMS Dock Appointments	WMS Dock Appointments object is to create new dock appointments or modify and delete the already existing dock appointments.
WMS_EPC_PUB	Electronic Product Code	It stores Electronic Product Codes such as GTIN, GID, SSCC, etc.
WMS_INSTALL	Warehouse Management System Installation Check	<p>This API has two purposes:</p> <ul style="list-style-type: none"> • 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.

Business Entity Code	Display Name	Description
WMS_REPLENISHMENT	Warehouse Replenishment	<p>This object provides access to the demand lines that are processed by replenishment code to facilitate post processing of the selected lines. For example, to back order a partially allocated replenishment move order.</p> <p>This object also allows to provide custom logic to select the list of demand lines which are to be replenished and back order the unselected demand lines.</p>
WMS_RFID_DEVICE	Warehouse Management Radio Frequency Identification	Warehouse Management Radio Frequency Identification Integration
WMS_RULES	Warehouse Rules Engine	<p>Rules engine object to provide support for custom logic that is honored during WMS rules engine execution.</p> <p>Custom strategy search, custom method to calculate the available capacity at a location, etc. can be handled via custom code.</p>
WMS_SHIPPING_TRANSACTION	Warehouse Management Shipping Transaction	Warehouse Management truck loading and shipping
WMS_TASKS	Warehouse Task Management	Support for warehouse task management like Query, Modify, Update, Split, Delete or Cancel the tasks.

Business Entity Code	Display Name	Description
WMS_WAVE_PLANNING	Warehouse Wave Planning	<p>Supports for wave planning to customize to:</p> <ul style="list-style-type: none"> • Add and remove lines from the waves being created based on customer's requirement • Raise wave exceptions based on the custom logic • Release the tasks in unreleased status based on custom logic
WMS_XDOCK	Warehouse Crossdocking	<p>Warehouse Crossdocking Integration can be used to customize the way crossdocking is done.</p> <p>Use custom method to:</p> <ul style="list-style-type: none"> • Decide crossdock criteria to be used. • Calculate the expected time. • Calculate the expected delivery time. • Sort the supply lines. • Sort the demand lines.
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	Display Name	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	Display Name	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_GROUPSALERSORDER	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	Display Name	Description
XTR_MARKET_DATA	Market Rate	Financial Market Rates Data
XTR_PAYMENT	XTR Payment	Treasury Payment represents the payments that are being made.
YMS_CUSTOM_LOGIC	Yard Custom Logic	Custom logic helps you to: <ul style="list-style-type: none"> • Determine the check-in location during yard check-in • Provide the item unit cost for a given item, organization, and document
YMS_DOCK_APPOINTMENTS	Yard Dock Appointments	A yard dock appointment object to create, modify, query, or delete dock appointments.
YMS_EQUIPMENT_ITEM	Yard Equipment Type	It indicates a yard equipment type that can be transacted in a yard organization.
YMS_INQUIRY	Yard Inquiry	It inquires about yard statistics, accessible organizations, equipment condition, equipment load status, etc.
YMS_TRANSACTIONS	Yard Transactions	Yard transactions can be like check-in, check-out, seal, unseal, and so on in a yard organization.
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>_bpel_<BPEL_PROCESS_NAME>.jar.
 - <prod>_bpel_<BPEL_PROCESS_NAME>.jar file should be placed under \$product_top/patch/115/jar/bpel.

- `<prod>_bpel_<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>_bpel_<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 `$product_top/patch/115/jar/bpel/<prod>_bpel_<BPEL_PROCESS_NAME>.wsdl`.
- WSDL File Name should not be changed from `<BPEL_PROCESS_NAME>.wsdl`.
- Standalone Parser should be run on annotated `$product_top/patch/115/jar/bpel/<prod>_bpel_<BPEL_PROCESS_NAME>.bpel/bpel<BPEL_PROCESS_NAME>.bpel`.
- `$product_top/patch/115/jar/bpel/<prod>_bpel_<BPEL_PROCESS_NAME>.bpel/bpel<BPEL_PROCESS_NAME>.ildt` should be loaded into Integration Repository.
- During the invocation 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-125
- `rep:scope`, page A-123

- rep:product, page A-124
- rep:category BUSINESS_ENTITY, page A-132

Optional Annotations

- link, page A-129
- see, page A-130
- rep:lifecycle, page A-127
- rep:compatibility, page A-128
- rep:ihelp, page A-131
- rep:metalink, page A-132

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/soapprovider/plsql/AR_INVOICE_API_PUB_2108/CREATE_SINGLE_INVOICE_1037
895/"
  xmlns:ns3="http://xmlns.oracle.
com/soapprovider/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"/>

```

```

<partnerLink 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	<p data-bbox="873 310 1369 369">Defines a user-friendly description of what the interface or method does.</p> <p data-bbox="873 401 1369 520">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.</p> <p data-bbox="873 552 1369 611">An example of a good beginning sentence could be as follows:</p> <p data-bbox="873 642 1369 722">"The Purchase Order Data Object holds the purchase order data including nested data objects such as lines and shipments."</p> <p data-bbox="873 753 1369 1003">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).</p> <p data-bbox="873 1035 1369 1062">A bad example would be: "Create an order."</p> <p data-bbox="873 1087 1369 1146">This description is barely usable. A better one would be:</p> <p data-bbox="873 1171 1369 1230">"Use this package to create a customer order, specifying header and line information."</p> <p data-bbox="873 1255 1369 1346">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:</p> <p data-bbox="873 1371 1369 1430">The following is an example on how to force a new line in the description:</p> <p data-bbox="873 1455 1369 1776">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.</p>

Annotation Type	<description sentence(s)>
Example	<pre> /*# * This is a sample description. Use * standard English capitalization and * punctuation. Write descriptions * carefully.</pre>
Required	Required for all interfaces that have <code>@rep:scope public</code> .
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	<p>Optionally, you can use the following HTML tags in your descriptions:</p> <pre> <body> <p> <h1> <h2> <h3></pre> <p><code><pre></code> for multiple code samples (should be enclosed by <code><code></code> tags)</p>

The following table describes the annotation information for `@rep:scope`:

`@rep:scope`

Annotation Type	<code>@rep:scope</code>
Syntax	<code>@rep:scope public private internal</code>
Usage	Indicates where to publish the interface, if at all.

Annotation Type	@rep:scope
Example	<p><code>@rep:scope public</code> means publish everywhere.</p> <p>Note: Public interfaces are displayed on the customer-facing UI.</p> <p><code>@rep:scope private</code> means that this interface is published to the Integration Repository but restricted for use by the owning team.</p> <p><code>@rep:scope internal</code> means publish within the company.</p>
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	<code>@rep:product StringShortCode</code>
Usage	Specifies the product shortname of the interface.
Example	<code>@rep:product PO</code>
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	<code>@rep:implementation StringClassName</code>
Usage	Specifies the implementation class name of the interface.
Example	<code>@rep:implementation oracle.apps. po.server.PurchaseOrdersAmImpl</code>
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	<code>@rep:displayname StringName</code>
Usage	Defines a user-friendly name for the interface.
Example	<code>@rep:displayname Purchase Order Summary</code>

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	<p data-bbox="971 306 1255 336">Display Name Guidelines</p> <p data-bbox="971 363 1442 455">These guidelines apply to display names for all technologies (interfaces, classes, methods, parameters, XMLG maps, and so on).</p> <p data-bbox="971 480 1390 539">Display names must meet the following criteria:</p> <ul data-bbox="971 564 1463 1339" style="list-style-type: none"> <li data-bbox="971 564 1442 623">• Be mixed case. Do not use all capitals or all lower case. <li data-bbox="971 665 1390 758">• Be singular rather than plural. For example, use "Customer" instead of "Customers". <li data-bbox="971 800 1430 858">• Be fully qualified and representative of your business area. <li data-bbox="971 900 1289 930">• Not have underscores (_). <li data-bbox="971 972 1284 1001">• Not end with a period (.). <li data-bbox="971 1043 1414 1073">• Not be the same as the internal name. <li data-bbox="971 1115 1463 1173">• Not begin with a product code or product name. <li data-bbox="971 1215 1463 1339">• 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. <p data-bbox="971 1375 1430 1535">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.</p> <p data-bbox="971 1560 1451 1682">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".</p>

The following table describes the annotation information for @rep:lifecycle:

@rep:lifecycle

Annotation Type	@rep:lifecycle
Syntax	<code>@rep:lifecycle active deprecated obsolete planned</code>
Usage	Indicates the lifecycle phase of the interface.
Example	<p><code>@rep:lifecycle active</code> means the interface is active.</p> <p><code>@rep:lifecycle deprecated</code> means the interface has been deprecated.</p> <p><code>@rep:lifecycle obsolete</code> means the interface is obsolete and must not be used.</p> <p><code>@rep:lifecycle planned</code> means the interface is planned for a future release. This is used for prototypes and mockups.</p>
Required	Optional.
Default	The default value is <code>active</code> .
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	<code>@rep:compatibility S N</code>

Annotation Type	<code>@rep:compatibility</code>
Usage	S indicates the lifecycle phase of the interface. N indicates that backward compatibility is not assured.
Example	<code>@rep:compatibility S</code>
Required	Optional.
Default	Conditional. The value is defaulted to S for <code>@rep:scope public</code> . 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	<code>@link</code>
	Note: This is supported only for a destination of Java.
Syntax	<code>{@link package.class#member label}</code>
Usage	Provides a link to another interface or method.
Example	<code>{@link #setAmounts(int,int,int,int) Set Amounts}</code>
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	<p>This is the standard Javadoc "@link" annotation, where the linked items are embedded as hyperlinks in the description that displays in the UI.</p> <p>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.</p>

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	<p>This is the standard Javadoc "@see" annotation.</p> <p>The linked items will display on the UI under a "See Also" heading.</p> <p>Usage in PL/SQL Code: @see package#procedure</p>

The following table describes the annotation information for @rep:ihelp:

@rep:ihelp

Annotation Type	@rep:ihelp
Syntax	<p>When used as a separate child annotation on a single line:</p> <pre>@rep:ihelp <product_shortname>/@<help_target> #<help_target> <link_text></pre> <p>When used as an inline annotation, add curly braces:</p> <pre>{@rep:ihelp <product_shortname>/@<help_target> #<help_target> <link_text>}</pre>
Usage	<p>Provides a link to an existing HTML online help page.</p> <p>product_shortname is the product short name.</p> <p>help_target is the help target that was manually embedded in the file by the technical writer, such as, "jtfacsum_jsp," "aolpo," "overview," "ast_aboutcollateral".</p> <p>For more information on how to customize Oracle E-Business Suite help, see Setting Up Oracle E-Business Suite Help, <i>Oracle E-Business Suite Setup Guide</i>.</p>
Example	<pre>@rep:ihelp #setAmounts(int,int, int,int)</pre>

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	<p>When used as a separate child annotation on a single line:</p> <pre>@rep:metalink <bulletin_number> <link_text></pre> <p>When used as an inline annotation, add curly braces:</p> <pre>{@rep:metalink <bulletin_number> <link_text>}</pre>
Usage	Provides a link to an existing My Oracle Support (formerly Oracle <i>MetaLink</i>) Knowledge Document.
Example	<pre>@rep:metalink 123456.1 See My Oracle Support Knowledge Document 123456.1</pre>
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	<ul style="list-style-type: none">• @rep:category BUSINESS_ENTITY BUSINESS_ENTITY_CODE• @rep:category IREP_CLASS_SUBTYPE JAVA_BEAN_SERVICES• @rep:category IREP_CLASS_SUBTYPE AM_SERVICES
Usage	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• @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. <p>See Business Entity Annotation Guidelines, page A-41 for additional details.</p>

Annotation Type	@rep:category
Required	<ul style="list-style-type: none"> 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 all Java interfaces that need 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	<p>Used when annotating concurrent programs to identify associated open interface tables or open interface views.</p> <p><table or view name> is the name of the table or view.</p> <p><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-132 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.</p> <p><direction flag> is optional and specifies one of the following: IN (default), OUT, or BOTH.</p>
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	<pre>@rep:standard <i>StringType</i> <i>StringVersionNumber</i> <i>StringSpecName</i></pre> <p>In the following example @rep:standard OAG 7.2 Process_PO_001 <i>StringType</i> is OAG, <i>StringVersionNumber</i> is 7.2 and <i>StringSpecName</i> is Process_PO_001</p> <p>See Annotation Syntax, page A-1 for details about this annotation's syntax.</p>

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 <code>@rep:standard RosettaNet 02.02.00 'Pip3B12-Shipping Order Confirmation'</code> , the <code>StringSpecName</code> is enclosed in Single Quotes because the spec name has empty spaces. It is not necessary to have these quotes if the <code>StringSpecName</code> does not have any empty spaces like the following example <code>@rep:standard RosettaNet 02.02.00 Pip3B12-PurchaseOrderConfirmation</code> .
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	<p><code>@rep:httpverb <HTTP_Method_Types></code></p> <p>Use a comma separated list of the HTTP verbs (GET and POST) at the method level.</p> <p>Note: GET and POST are the supported HTTP methods for Java Bean Services and Application Module Services in this release.</p>

Annotation Type	@rep:httpverb
Usage	<p>Use this annotation to indicate the HTTP Verbs suitable for the current method or operation.</p> <p>If a method is not annotated with <code>POST httpverb</code>, the POST checkbox 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 <code>GET httpverb</code>, then the GET checkbox becomes inactive or disabled for that method. This means that method will not be exposed as REST service with GET operation.</p> <p>This annotation is available for Java files only.</p>
Example	<p>The comma separate list can be used in the following ways:</p> <pre>@rep:httpverb get @rep:httpverb post @rep:httpverb get, post</pre>
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	<code>@rep:interface StringClassName</code> where the <code>StringClassName</code> syntax is <code>transactiontype: subtype</code> . 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 <code>StringClassName</code> is always <code>transactiontype:subtype</code> <code>@rep:interface PO:POC</code>
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	<code>@param paramName paramDescription</code> 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	<code>@param PONumber The purchase order number.</code>
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 <i>StringDescription</i>
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	<p data-bbox="724 310 911 338"><code>rep:paraminfo</code></p> <p data-bbox="724 359 1333 449">The <code>rep:paraminfo</code> annotation must come immediately in the line following the parameter's <code>@param</code> or <code>@return</code> annotation it is describing.</p> <p data-bbox="724 470 911 497"><code>rep:innertype</code></p> <p data-bbox="724 518 1308 588">Optional inline annotation to describe the inner type of generic objects such as collections.</p> <p data-bbox="724 609 911 636"><code>rep:precision</code></p> <p data-bbox="724 657 1268 726">Optional inline annotation to specify the parameter precision. Used for Strings and numbers.</p> <p data-bbox="724 747 894 774"><code>rep:required</code></p> <p data-bbox="724 795 1365 865">Optional inline annotation to indicate that a not null must be supplied. This is only needed for non-PL/SQL technologies.</p> <p data-bbox="724 886 911 913"><code>rep:key_param</code></p> <p data-bbox="724 934 1365 1100">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 <code>rep:required</code> must be present. Additionally, this annotation is not applicable to <code>rep:paraminfo</code> after <code>@return</code> annotation.</p>
Example	<pre data-bbox="724 1150 1341 1688">/** * 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);</pre>
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	<code>@rep:direction <OUT IN></code>
Usage	Indicates whether the interface is outbound or inbound.
Example	<code>@rep:direction OUT</code>
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	<code>@rep:service</code>
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	<pre> /** * 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 */ </pre>
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	<pre> /** * 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 */ </pre>
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	<code>@rep:appscontext <NONE, APPL, RESP, USER, NLS, or ORG></code>
Usage	Specifies the context required to run the method.
Example	<code>@rep:appscontext USER</code>
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	<code>@rep:comment <comment></code>
Usage	This annotation is skipped by the parsers. It is for use by product teams when a non-published comment is desired.
Example	<code>@rep:comment This is a sample comment.</code>

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	<p>To indicate the E-Commerce Gateway maps that are associated with a concurrent program.</p> <p><map_name> where map_name is the default map name.</p> <p><sequence_number> is an integer used to tell the UI the display order of the different pieces.</p>
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.

Configuring Server Connection

Application Server Connection

Security is the most critical feature to guard service content from unauthorized access. To protect the SOA composite applications and other resources deployed in the Oracle SOA Suite WebLogic Server domain and ensure that service invocations are successfully processed, necessary server connection need to be performed.

This section includes the following server connection:

- [Creating an Application Server Connection](#), page B-1

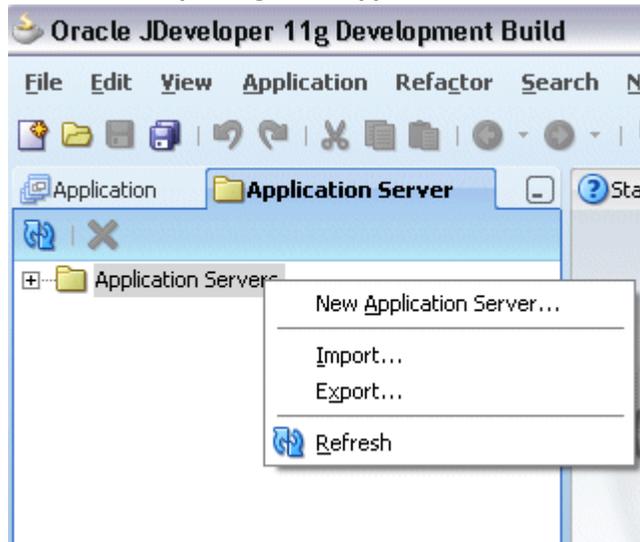
Creating an Application Server Connection

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 (Oracle WebLogic Server) connection.

Use the following steps to create the connection to an Oracle WebLogic Server:

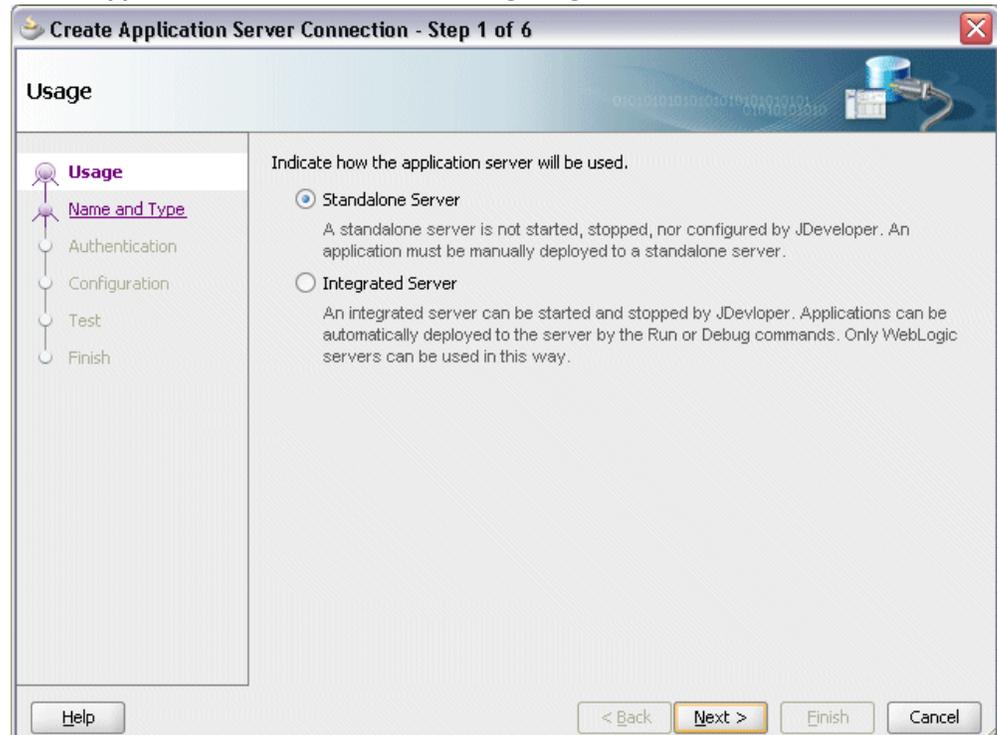
1. In Oracle JDeveloper, select **View >Application Server Navigator** to open the Application Server tab.

Oracle JDeveloper Page with Application Server Tab



2. Right-click on the Application Server and select **New Application Server** to open the Create Application Server Connection wizard.
3. In the Usage page, select the **Standalone Server** radio button and click **Next**.

Create Application Server Connection - Usage Page



4. Enter the connection name (such as 'soa-server1') and select **WebLogic 10.3** as the connection type.

Create Application Server Connection - Name and Type Page

The screenshot shows a dialog box titled "Create Application Server Connection - Step 2 of 6". The main heading is "Name and Type". On the left, there is a vertical navigation pane with the following steps: Usage, Name and Type (highlighted), Authentication, Configuration, Test, and Finish. The main area contains the following text and controls:

Specify a unique name and type for the connection. The name must be a valid Java identifier.

Create connection in: Resource Palette

Connection Name:

Connection Type:

At the bottom, there are four buttons: Help, < Back, Next >, and Cancel.

Click **Next**.

5. Enter a valid username (such as `weblogic`) and the password specified during Oracle SOA Suite installation. Click **Next**.

Create Application Server Connection - Authentication Page

Create Application Server Connection - Step 3 of 6

Authentication

Specify a username and password to authenticate the connection.

Usage
Name and Type
Authentication
Configuration
Test
Finish

Username:
weblogic

Password:

Help < Back Next > Finish Cancel

6. Enter the WebLogic Server connection host name and port information. In the Weblogic Domain field, enter 'soainfra'.

Create Application Server Connection - Configuration Page

WebLogic Server connections use a host name and port to establish a connection. The Domain of the target will be verified

Weblogic Hostname (Administration Server):
localhost

Port: 7001 SSL Port: 7002

Always use SSL

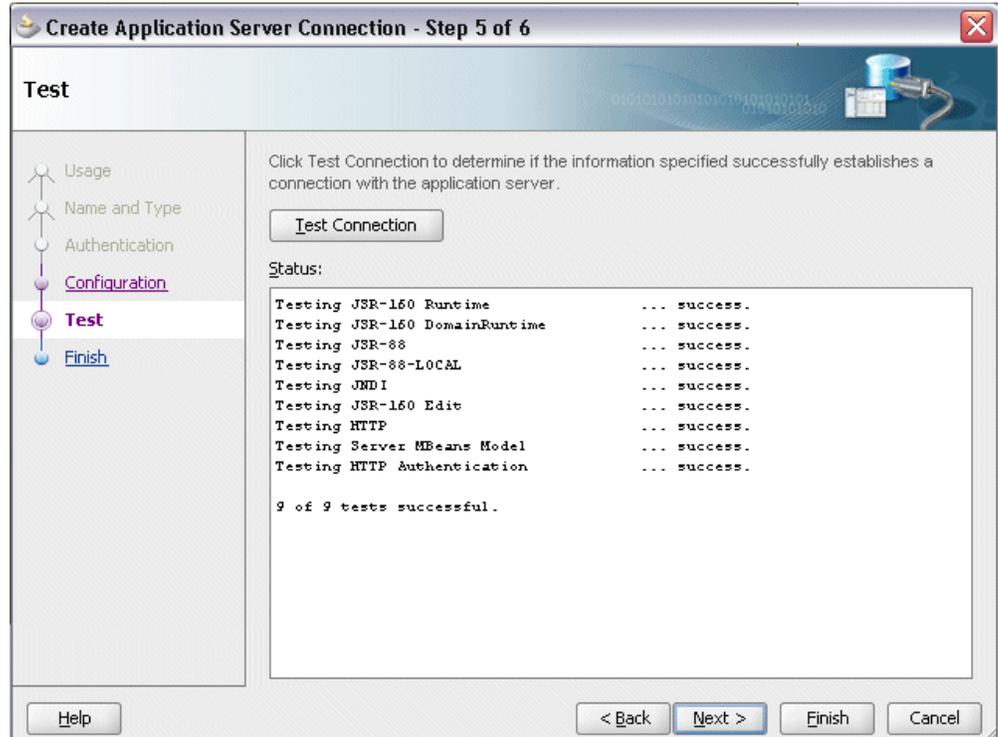
Weblogic Domain:
soainfra

Help < Back Next > Finish Cancel

Click **Next**. The Test page is displayed.

7. Click **Test Connection** to validate your server configuration. You should find success messages populated in the Status window.

Create Application Server Connection - Test Page



Click **Finish**.

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/soapprovider/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>xxxxxxxx</cre:REQUESTOR_ID>
      <cre:ASSIGNEE_RESOURCE_ID>xxxxxxxx</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>xxxxxxxx</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_ITEM>
      <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_ITEM>
    </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' encoding = 'UTF-8' standalone = 'no'?>
<!-- Oracle eXtensible Markup Language Gateway Server -->
<!DOCTYPE PROCESS_PO_007 SYSTEM "003_process_po_007.dtd">
  <PROCESS_PO_007>
    <CNTROLAREA>
      <BSR>
        <VERB>PROCESS</VERB>
        <NOUN>PO</NOUN>
      <REVISION>007</REVISION>
    </BSR>
    <SENDER>
      <LOGICALID>ORACLE</LOGICALID>
      <COMPONENT>PURCHASING</COMPONENT>
      <TASK>POISSUE</TASK>
      <REFERENCEID>MO4YD220.US.EXAMPLE.COM:oracle.apps.po.event.xmlpo:
32636-148970</REFERENCEID>
      <CONFIRMATION>0</CONFIRMATION>
      <LANGUAGE>us</LANGUAGE>
      <CODEPAGE>UTF8</CODEPAGE>
      <AUTHID>APPS</AUTHID>
    </SENDER>
    <DATETIME qualifier="CREATION" type="T" index="1">
      <YEAR>2011</YEAR>
      <MONTH>04</MONTH>
      <DAY>05</DAY>
      <HOUR>23</HOUR>
      <MINUTE>44</MINUTE>
      <SECOND>09</SECOND>
      <SUBSECOND>0000</SUBSECOND>
      <TIMEZONE>+0000</TIMEZONE>
    </DATETIME>
    <OPERAMT qualifier="EXTENDED" type="T">
      <VALUE>1329432</VALUE>
      <NUMOFDEC>2</NUMOFDEC>
      <SIGN>+</SIGN>
      <CURRENCY>USD</CURRENCY>
      <UOMVALUE>1</UOMVALUE>
      <UOMNUMDEC>0</UOMNUMDEC>
      <UOM/>
    </OPERAMT>
    <POID>PO-4466-5</POID>
    <POTYPE>STANDARD</POTYPE>
    <ACKREQUEST>N</ACKREQUEST>
    <CONTRACTB/>
    <CONTRACTS/>
    <DESCRIPTN/>
    <PORELEASE/>
    <USERAREA><DATETIME qualifier="ACTSTART" type="T" index="1"
><YEAR/><MONTH/><DAY/><HOUR/><MINUTE/><SECOND/><SUBSECOND/><TIMEZONE/></
DATETIME><DATETIME qualifier="ACTEND" type="T" index="1"
><YEAR/><MONTH/><DAY/><HOUR/><MINUTE/><SECOND/><SUBSECOND/><TIMEZONE/></
DATETIME><DATETIME><FOB><DESCRIPTN>Vendor's responsibility ceases upon
transfer to
carrier</DESCRIPTN><TERMID>Origin</TERMID></FOB><TANDC/><FTTERM><DESCRIP
TN>Buyer pays
freight</DESCRIPTN><TERMID>Due</TERMID></FTTERM><EXCHRATE/><DATETIME
qualifier="EXCHRATE"
><YEAR>2011</YEAR><MONTH>04</MONTH><DAY>05</DAY><HOUR>00</HOUR><MINUTE>0
0</MINUTE><SECOND>00</SECOND><SUBSECOND>0000</SUBSECOND><TIMEZONE>+0000<
/TIMEZONE></DATETIME><DATETIME ualifier="APPREQ" type="T" index="1"
><YEAR/><MONTH/><DAY/><HOUR/><MINUTE/><SECOND/><SUBSECOND/><TIMEZONE/></
DATETIME><CONFIRM>N</CONFIRM><SHIPPINGCONTROL/><DFFPOHEADER><ATTRIBUTE1/
><ATTRIBUTE2/><ATTRIBUTE3/><ATTRIBUTE4/><ATTRIBUTE5/><ATTRIBUTE6/><ATTRI
BUTE7/><ATTRIBUTE8/><ATTRIBUTE9/><ATTRIBUTE10/><ATTRIBUTE11/><ATTRIBUTE1
2/><ATTRIBUTE13/><ATTRIBUTE14/><ATTRIBUTE15/><ATTRIBUTE16/></DFFPOHEADER
><PCARDHDR><MEMBERNAME/><PCARDNUM>0</PCARDNUM><DATETIME qualifier="

```

EXPIRATION"
 ><YEAR>2011</YEAR><MONTH>04</MONTH><DAY>05</DAY><HOUR>00</HOUR><MINUTE>0
 0</MINUTE><SECOND>00</SECOND><SUBSECOND>0000</SUBSECOND><TIMEZONE>+0000<
 /TIMEZONE></DATETIME><PCARDBRAND/></PCARDHDR></USERAREA>
 <PARTNER>
 <NAME index="1">Example Inc.</NAME>
 <ONETIME>0</ONETIME>
 <PARTNRID><PARTNRID/>
 <PARTNRTYPE>Supplier</PARTNRTYPE>
 <CURRENCY>USD</CURRENCY>
 <DUNSNUMBER/>
 <PARTNRIDX>Example-01</PARTNRIDX>
 <TAXID/>
 <TERMINID/>

<USERAREA><DFVENDOR><ATTRIBUTE1/><ATTRIBUTE2/><ATTRIBUTE3/><ATTRIBUTE4/
 ><ATTRIBUTE5/><ATTRIBUTE6/><ATTRIBUTE7/><ATTRIBUTE8/><ATTRIBUTE9/><ATTRI
 BUTE10/><ATTRIBUTE11/><ATTRIBUTE12/><ATTRIBUTE13/><ATTRIBUTE14/><ATTRIBU
 TE15/><ATTRIBUTE16/></DFVENDOR><CUSTOMERNUM/></USERAREA>
 <ADDRESS>
 <ADDRLINE index="1"></ADDRLINE>
 <ADDRLINE index="2"/>
 <ADDRLINE index="3"/>
 <ADDRTYPE/>
 <CITY></CITY>
 <COUNTRY></COUNTRY>
 <COUNTRY/>
 <DESCRIPTN></DESCRIPTN>
 <FAX index="1"/>
 <POSTALCODE></POSTALCODE>
 <REGION/>
 <STATEPROVN></STATEPROVN>
 <TAXJRSRCTN/>
 <TELEPHONE index="1"></TELEPHONE>
 <TELEPHONE index="2"/>
 <TELEPHONE index="3"/>
 <URL/>

<USERAREA><DFVENDORSITE><ATTRIBUTE1/><ATTRIBUTE2/><ATTRIBUTE3/><ATTRIBU
 TE4/><ATTRIBUTE5/><ATTRIBUTE6/><ATTRIBUTE7/><ATTRIBUTE8/><ATTRIBUTE9/><A
 TTRIBUTE10/><ATTRIBUTE11/><ATTRIBUTE12/><ATTRIBUTE13/><ATTRIBUTE14/><ATT
 RIBUTE15/><ATTRIBUTE16/></DFVENDORSITE></USERAREA>
 </ADDRESS>
 <CONTACT>
 <NAME index="1"/>
 <EMAIL/>
 <FAX index="1"></FAX>
 <TELEPHONE index="1"></TELEPHONE>
 </CONTACT>
 </PARTNER>
 <PARTNER>
 <NAME index="1">Example Inc.</NAME>
 <ONETIME>0</ONETIME>
 <PARTNRID>xxx<PARTNRID/>
 <PARTNRTYPE>SoldTo</PARTNRTYPE>
 <CURRENCY>USD</CURRENCY>
 <DUNSNUMBER/>
 <PARTNRIDX>Example-01</PARTNRIDX>
 <PAYMETHOD/>
 <TAXID/>
 <TERMINID/>
 <USERAREA/>
 <ADDRESS>
 </ADDRESS>
 <CONTACT>
 <NAME index="1">xxxxxx, Ms. xxxxxx</NAME>

```

<CONTACTTYPE/>
  <DESCRIPTN/>
  <EMAIL>xxxxxxx@example.com</EMAIL>
  <FAX index="1"/>
  <TELEPHONE index="1"/>
  <USERAREA/>
</CONTACT>
</PARTNER>
<PARTNER>
  <NAME index="1">Example Inc.</NAME>
  <ONETIME>0</ONETIME>
  <PARTNRID>xxx</PARTNRID/>
  <PARTNRTYPE>BillTo</PARTNRTYPE>
  <CURRENCY/>
  <DUNSNUMBER/>
  <PARTNRIDX>Example-01</PARTNRIDX>
  <PAYMETHOD/>
  <TERMID/>
  <USERAREA/>
  <ADDRESS>
  </ADDRESS>
</PARTNER>
<PARTNER>
  <NAME index="1">UPS</NAME>
  <ONETIME>0</ONETIME>
  <PARTNRID>xxx</PARTNRID/>
  <PARTNRTYPE>Carrier</PARTNRTYPE>
  <PARTNRIDX>UPS</PARTNRIDX>
</PARTNER>
<POTERM>
  <DESCRIPTN>Scheduled for payment 30 days from the invoice date
(invoice terms date = system date, goods received date, invoice date or
invoice received date). Invoice terms date can default from supplier
header, site, PO, system default, etc.</DESCRIPTN>
  <TERMID>30 Net (terms date + 30)</TERMID>
  <DAYSNUM/>
  <QUANTITY qualifier="PERCENT">
    <VALUE/>
    <NUMOFDEC/>
    <SIGN/>
    <UOM/>
  </QUANTITY>
  <USERAREA/>
</POTERM>
</POORDERHDR>
<POORDERLIN>
  <QUANTITY qualifier="ORDERED">
    <VALUE>12</VALUE>
    <NUMOFDEC/>
    <SIGN>+</SIGN>
    <UOM>Ea</UOM>
  </QUANTITY>
  <OPERAMT qualifier="UNIT" type="T">
    <VALUE>110786</VALUE>
    <NUMOFDEC>2</NUMOFDEC>
    <SIGN>+</SIGN>
    <CURRENCY>USD</CURRENCY>
    <UOMVALUE>1</UOMVALUE>
    <UOMNUMDEC>0</UOMNUMDEC>
    <UOM>Ea</UOM>
  </OPERAMT>
  <POLINENUM>1</POLINENUM>
  <HAZRDMATL/>
  <ITEMRV></ITEMRV>
  <ITEMRVX/>
  <POLNSTATUS/>

```

```

<DESCRIPTN></DESCRIPTN>
  <ITEM>AS54888</ITEM>
  <ITEMX/>
  <USERAREA><REQUESTOR/><CATEGORYID>PRODUCTN.
FINGOODS</CATEGORYID><CONTRACTNUM/><CONTRACTPONUM/><CONTRACTPOLINENUM/><
VENDORQUOTENUM/><CONFIGID/><LISTPRICE>1107.
86</LISTPRICE><MARKETPRICE>0</MARKETPRICE><PRICENOTTOEXCEED/><NEGPRICE>N
</NEGPRICE><TAXABLE>N</TAXABLE><TXNREASONCODE/><TYPE109>AVAN/N</TYPE109
9><LINEORDERTYPE>Goods</LINEORDERTYPE><HAZRDUNNUM/><HAZRDUNDESC/><DFFLIN
E><ATTRIBUTE1/><ATTRIBUTE2/><ATTRIBUTE3/><ATTRIBUTE4/><ATTRIBUTE5/><ATTR
IBUTE6/><ATTRIBUTE7/><ATTRIBUTE8/><ATTRIBUTE9/><ATTRIBUTE10/><ATTRIBUTE1
1/><ATTRIBUTE12/><ATTRIBUTE13/><ATTRIBUTE14/><ATTRIBUTE15/><ATTRIBUTE16/
></DFFLINE><DFITEM><ATTRIBUTE1/><ATTRIBUTE2/><ATTRIBUTE3/><ATTRIBUTE4/>
<ATTRIBUTE5/><ATTRIBUTE6/><ATTRIBUTE7/><ATTRIBUTE8/><ATTRIBUTE9/><ATTRIB
UTE10/><ATTRIBUTE11/><ATTRIBUTE12/><ATTRIBUTE13/><ATTRIBUTE14/><ATTRIBUT
E15/><ATTRIBUTE16/></DFITEM><KFFITEM><ATTRIBUTE1>PRODUCTN</ATTRIBUTE1><
ATTRIBUTE2>FINGOODS</ATTRIBUTE2><ATTRIBUTE3/><ATTRIBUTE4/><ATTRIBUTE5/><
ATTRIBUTE6/><ATTRIBUTE7/><ATTRIBUTE8/><ATTRIBUTE9/><ATTRIBUTE10/><ATTRIB
UTE11/><ATTRIBUTE12/><ATTRIBUTE13/><ATTRIBUTE14/><ATTRIBUTE15/><ATTRIBUT
E16/><ATTRIBUTE17/><ATTRIBUTE18/><ATTRIBUTE19/><ATTRIBUTE20/></KFFITEM><
GLOBALCONTRACT/><GLOBALCONTRACTLIN/><JOBTITLE/><AMOUNT qualifier="TOTAL"
type="T"
><VALUE/><NUMOFDEC/><SIGN/><CURRENCY/><DRCR>C</DRCR></AMOUNT><CONTRACTOR
FIRSTNAME/><CONTRACTORLASTNAME/><DATETIME qualifier="ACTSTART" type="T"
index="1"
><YEAR/><MONTH/><DAY/><HOUR/><MINUTE/><SECOND/><SUBSECOND/><TIMEZONE/></
DATETIME><DATETIME qualifier="ACTEND" type="T" index="1"
><YEAR/><MONTH/><DAY/><HOUR/><MINUTE/><SECOND/><SUBSECOND/><TIMEZONE/></
DATETIME></USERAREA>
  <POLINESCHD>
  <DATETIME qualifier="NEEDELV" type="T" index="1">
  <YEAR>2011</YEAR>
  <MONTH>04</MONTH>
  <DAY>21</DAY>
  <HOUR>00</HOUR>
  <MINUTE>00</MINUTE>
  <SECOND>00</SECOND>
  <SUBSECOND>0000</SUBSECOND>
  <TIMEZONE>+0000</TIMEZONE>
  </DATETIME>
  <QUANTITY qualifier="ORDERED">
  <VALUE>12</VALUE>
  <NUMOFDEC/>
  <SIGN>+</SIGN>
  <UOM>Ea</UOM>
  </QUANTITY>
  <DESCRIPTN/>
  <PSCLINENUM>1</PSCLINENUM>
  <USERAREA><DATETIME qualifier="PROMSHIP"
><YEAR/><MONTH/><DAY/><HOUR/><MINUTE/><SECOND/><SUBSECOND/><TIMEZONE/></
DATETIME><DATETIME qualifier="APPROVAL"
><YEAR/><MONTH/><DAY/><HOUR/><MINUTE/><SECOND/><SUBSECOND/><TIMEZONE/></
DATETIME><OPERAMT qualifier="UNIT" type="T"
><VALUE>110786</VALUE><NUMOFDEC>2</NUMOFDEC><SIGN>+</SIGN><CURRENCY>USD<
/CURRENCY><UOMVALUE>1</UOMVALUE><UOMNUMDEC>0</UOMNUMDEC><UOM>Ea</UOM></O
PERAMT><PRICEOVRD/><TAXABLE>N</TAXABLE><TAXCODE/><PARTNER><NAME index="
1">Example Inc.
</NAME><ONETIME>0</ONETIME><PARTNRID>xxx</PARTNRID><PARTNRTYPE>ShipTo</P
ARTNRTYPE><CURRENCY>USD</CURRENCY><DUNSNUMBER/><PARTNRIDX>Example-
01</PARTNRIDX><PAYMETHOD/><TERMID/><USERAREA/>
  <ADDRESS></ADDRESS>
  <CONTACT><NAME index="1"/><CONTCCTTYPE/><EMAIL/><FAX index="1"
/><TELEPHONE index="1"/></CONTACT>
  </PARTNER>
  <PARTNER><NAME index="1"
/><ONETIME/><PARTNRID/><PARTNRTYPE>DeliveryTo</PARTNRTYPE><PARTNRIDX/><U

```

```

SERAREA/><ADDRESS><ADDRLINE index="1"/><ADDRLINE index="2"
/><ADDRTYPE/><ADDRESS><CONTACT><NAME index="1"
/><CONCTTYPE/><EMAIL/><FAX index="1"/><TELEPHONE index="1"/>
</CONTACT></PARTNER><DROPSHIPDETAILS><DROPSHIPMENT/><DROPSHIPCUSTNAME/><
SHIPINSTR/><PACKINSTR/><SHIPMETHOD/><CUSTOMERPONUM/><CUSTOMERLINENUM/><C
USTOMERSHIPNUM/><CUSTOMERDESC/></DROPSHIPDETAILS><CONSIGNEDINV>N</CONSIG
NEDINV><DISTPROJECT><REQUESTOR/><DISTNUM>1</DISTNUM><PROJECTNUM/><PROJEC
TTYPE/><TASKNUM/><QUANTITY qualifier="ORDERED"
><VALUE>12</VALUE><NUMOFDEC/><SIGN>+</SIGN><UOM>Ea</UOM></QUANTITY><CONV
RATE/><DATETIME qualifier="EXCHRATEDATE"
><YEAR>2011</YEAR><MONTH>04</MONTH><DAY>05</DAY><HOUR>00</HOUR><MINUTE>0
0</MINUTE><SECOND>00</SECOND><SUBSECOND>0000</SUBSECOND><TIMEZONE>+0000<
/TIMEZONE></DATETIME><DESTTYPE>INVENTORY</DESTTYPE><DFFDISTRIBUTN><ATTRI
BUTE1/><ATTRIBUTE2/><ATTRIBUTE3/><ATTRIBUTE4/><ATTRIBUTE5/><ATTRIBUTE6/>
<ATTRIBUTE7/><ATTRIBUTE8/><ATTRIBUTE9/><ATTRIBUTE10/><ATTRIBUTE11/><ATTR
IBUTE12/><ATTRIBUTE13/><ATTRIBUTE14/><ATTRIBUTE15/><ATTRIBUTE16/></DFFDI
STRIBUTN></DISTPROJECT><AMOUNT qualifier="TOTAL" type="T"
><VALUE/><NUMOFDEC/><SIGN/><CURRENCY/><DRCR>C</DRCR></AMOUNT></USERAREA>

    </POLINESCHD>
  </POORDERLIN>
<POORDERLIN>
  <QUANTITY qualifier="ORDERED">
    <VALUE>13</VALUE>
    <NUMOFDEC/>
    <SIGN>+</SIGN>
    <UOM>Ea</UOM>
  </QUANTITY>
  <OPERAMT qualifier="UNIT" type="T">
    <VALUE>110786</VALUE>
    <NUMOFDEC>2</NUMOFDEC>
    <SIGN>+</SIGN>
    <CURRENCY>USD</CURRENCY>
    <UOMVALUE>1</UOMVALUE>
    <UOMNUMDEC>0</UOMNUMDEC>
    <UOM>Ea</UOM>
  </OPERAMT>
  <POLINENUM>2</POLINENUM>
  <HAZRDMATL/>
  <ITEMRV></ITEMRV>
  <ITEMRVX/>
  <POLNSTATUS/>
  <DESCRIPTN></DESCRIPTN>
  <ITEM>AS00021</ITEM>
  <ITEMX/>
  <USERAREA><REQUESTOR/><CATEGORYID>PRODUCTN.
FINGOODS</CATEGORYID><CONTRACTNUM/><CONTRACTPONUM/><CONTRACTPOLINENUM/><
VENDORQUOTENUM/><CONFIGID/><LISTPRICE>1107.
86</LISTPRICE><MARKETPRICE>0</MARKETPRICE><PRICENOTTOEXCEED/><NEGPRICE>N
</NEGPRICE><TAXABLE>N</TAXABLE><TXNREASONCODE/><TYPE1099>AVAN/N</TYPE109
9><LINEORDERTYPE>Goods</LINEORDERTYPE><HAZRDUNNUM/><HAZRDUNDESC/><DFFLIN
E><ATTRIBUTE1/><ATTRIBUTE2/><ATTRIBUTE3/><ATTRIBUTE4/><ATTRIBUTE5/><ATTR
IBUTE6/><ATTRIBUTE7/><ATTRIBUTE8/><ATTRIBUTE9/><ATTRIBUTE10/><ATTRIBUTE1
1/><ATTRIBUTE12/><ATTRIBUTE13/><ATTRIBUTE14/><ATTRIBUTE15/><ATTRIBUTE16/
></DFFLINE><DFFITEM><ATTRIBUTE1/><ATTRIBUTE2/><ATTRIBUTE3/><ATTRIBUTE4/>
<ATTRIBUTE5/><ATTRIBUTE6/><ATTRIBUTE7/><ATTRIBUTE8/><ATTRIBUTE9/><ATTRIB
UTE10/><ATTRIBUTE11/><ATTRIBUTE12/><ATTRIBUTE13/><ATTRIBUTE14/><ATTRIBUT
E15/><ATTRIBUTE16/></DFFITEM><KFFITEM><ATTRIBUTE1>PRODUCTN</ATTRIBUTE1><
ATTRIBUTE2>FINGOODS</ATTRIBUTE2><ATTRIBUTE3/><ATTRIBUTE4/><ATTRIBUTE5/><
ATTRIBUTE6/><ATTRIBUTE7/><ATTRIBUTE8/><ATTRIBUTE9/><ATTRIBUTE10/><ATTRIB
UTE11/><ATTRIBUTE12/><ATTRIBUTE13/><ATTRIBUTE14/><ATTRIBUTE15/><ATTRIBU
TE16/><ATTRIBUTE17/><ATTRIBUTE18/><ATTRIBUTE19/><ATTRIBUTE20/></KFFITEM><
GLOBALCONTRACT/><GLOBALCONTRACTLIN/><JOBTTITLE/><AMOUNT qualifier="TOTAL"
type="T"
><VALUE/><NUMOFDEC/><SIGN/><CURRENCY/><DRCR>C</DRCR></AMOUNT><CONTRACTOR
FIRSTNAME/><CONTRACTORLASTNAME/><DATETIME qualifier="ACTSTART" type="T"

```

```

index="1"
><YEAR/><MONTH/><DAY/><HOUR/><MINUTE/><SECOND/><SUBSECOND/><TIMEZONE/></
DATETIME><DATETIME qualifier="ACTEND" type="T" index="1"
><YEAR/><MONTH/><DAY/><HOUR/><MINUTE/><SECOND/><SUBSECOND/><TIMEZONE/></
DATETIME></USERAREA>
  <POLINESCHD>
    <DATETIME qualifier="NEEDELV" type="T" index="1">
      <YEAR>2011</YEAR>
      <MONTH>04</MONTH>
      <DAY>21</DAY>
      <HOUR>00</HOUR>
      <MINUTE>00</MINUTE>
      <SECOND>00</SECOND>
      <SUBSECOND>0000</SUBSECOND>
      <TIMEZONE>+0000</TIMEZONE>
    </DATETIME>
    <QUANTITY qualifier="ORDERED">
      <VALUE>12</VALUE>
      <NUMOFDEC/>
      <SIGN>+</SIGN>
      <UOM>Ea</UOM>
    </QUANTITY>
    <DESCRIPTN/>
    <PSCLINENUM>1</PSCLINENUM>
    <USERAREA><DATETIME qualifier="PROMSHIP"
  ><YEAR/><MONTH/><DAY/><HOUR/><MINUTE/><SECOND/><SUBSECOND/><TIMEZONE/></
  DATETIME><DATETIME qualifier="APPROVAL"
  ><YEAR/><MONTH/><DAY/><HOUR/><MINUTE/><SECOND/><SUBSECOND/><TIMEZONE/></
  DATETIME><OPERAMT qualifier="UNIT" type="T"
  ><VALUE>110786</VALUE><NUMOFDEC>2</NUMOFDEC><SIGN>+</SIGN><CURRENCY>USD<
  /CURRENCY><UOMVALUE>1</UOMVALUE><UOMNUMDEC>0</UOMNUMDEC><UOM>Ea</UOM></O
  PERAMT><PRICEOVRD/><TAXABLE>N</TAXABLE><TAXCODE/><PARTNER><NAME index="
  1">Example Inc.
  </NAME><ONETIME>0</ONETIME><PARTNRID>xxx</PARTNRID><PARTNRTYPE>ShipTo</P
  ARTNRTYPE><CURRENCY>USD</CURRENCY><DUNSNUMBER/><PARTNRIDX>Example-
  01</PARTNRIDX><PAYMETHOD/><TERMID/></USERAREA/>
    <ADDRESS></ADDRESS>
    <CONTACT><NAME index="1"/><CONTCTTYPE/><EMAIL/><FAX index="1"
  /><TELEPHONE index="1"/></CONTACT>
    </PARTNER>
    <PARTNER><NAME index="1"
  /><ONETIME/><PARTNRID/><PARTNRTYPE>DeliveryTo</PARTNRTYPE><PARTNRIDX/><U
  SERAREA/><ADDRESS><ADDRLINE index="1"/><ADDRLINE index="2"
  /><ADDRTYPE/></ADDRESS><CONTACT><NAME index="1"
  /><CONTCTTYPE/><EMAIL/><FAX index="1"/><TELEPHONE index="1"
  /></CONTACT></PARTNER><DROPSHIPDETAILS><DROPSHIPMENT/><DROPSHIPCUSTNAME/
  ><SHIPINSTR/><PACKINSTR/><SHIPMETHOD/><CUSTOMERPONUM/><CUSTOMERLINENUM/>
  <CUSTOMERSHIPNUM/><CUSTOMERDESC/></DROPSHIPDETAILS><CONSIGNEDINV>N</CONS
  IGNEDINV><DISTPROJECT><REQUESTOR/><DISTNUM>1</DISTNUM><PROJECTNUM/><PROJ
  ECTTYPE/><TASKNUM/><QUANTITY qualifier="ORDERED"
  ><VALUE>12</VALUE><NUMOFDEC/><SIGN>+</SIGN><UOM>Ea</UOM></QUANTITY><CONV
  RATE/><DATETIME qualifier="EXCHRATEDATE"
  ><YEAR>2011</YEAR><MONTH>04</MONTH><DAY>05</DAY><HOUR>00</HOUR><MINUTE>0
  0</MINUTE><SECOND>00</SECOND><SUBSECOND>0000</SUBSECOND><TIMEZONE>+0000<
  /TIMEZONE></DATETIME><DESTTYPE>INVENTORY</DESTTYPE><DFFDISTRIBUTN><ATTRI
  BUTE1/><ATTRIBUTE2/><ATTRIBUTE3/><ATTRIBUTE4/><ATTRIBUTE5/><ATTRIBUTE6/>
  <ATTRIBUTE7/><ATTRIBUTE8/><ATTRIBUTE9/><ATTRIBUTE10/><ATTRIBUTE11/><ATTR
  IBUTE12/><ATTRIBUTE13/><ATTRIBUTE14/><ATTRIBUTE15/><ATTRIBUTE16/></DFFDI
  STRIBUTN></DISTPROJECT><AMOUNT qualifier="TOTAL" type="T"
  ><VALUE/><NUMOFDEC/><SIGN/><CURRENCY/><DRCR>C</DRCR></AMOUNT></USERAREA>

  </POLINESCHD>
  </POORDERLIN>
  </PROCESS_PO>
  </DATAAREA>
  </PROCESS_PO_007>

```

Glossary

Agent

A named point of communication within a system.

Agent Listener

A type of service component that processes event messages on inbound agents.

Asynchronous Operation

Unlike the synchronous service operations that obtain the result immediately, asynchronous operations may require a significant amount of time to process a request.

However, the client that invoked the Oracle E-Business Suite web service can continue with other processing in the meantime rather than wait for the response.

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.

Callback Pattern

Callback pattern is an important communication method in asynchronous services. An asynchronous callback means that a request is made to the service provider and a response (callback) is sent back to the requester when it is ready. This pattern can be used in conjunction with acknowledgement to recognize the receipt of a request sent by a requester.

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.

Integration Repository Parser

It is a standalone design-time tool used by the integration administrator to validate annotated custom interface definitions against the annotation standards and generate an Integration Repository loader file (iLDT). This generated iLDT file can be uploaded to Integration Repository where custom interfaces can be exposed to all users.

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 predefined 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.

Service Monitor

It is the monitoring and auditing tool in Oracle E-Business Suite allowing administrators to monitor inbound SOAP and REST service invocation messages.

It is known as SOA Monitor in earlier releases.

Service Invocation Monitor

Service Invocation Monitor is a monitoring and auditing tool allowing administrators to monitor outbound SOAP and REST service invocations from Oracle E-Business Suite through Service Invocation 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.

Synchronous Operation

Synchronous operation provides an immediate response to a query. In this situation, the client connection remains open from the time the request is submitted to the server. The client will wait until the server sends back the response message.

Username Token

A type of security model based on username and password to authenticate SOAP requests at runtime.

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 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 invokes 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

B

- basic information about the REST service invocation is captured as REST service metadata:
 - Overview, 13-1
- business events
 - annotate, 11-19
 - download, 11-16
 - Upload file to the database, 11-22
 - Upload iLDT files, 11-23
 - validate, 11-21
- Business Service Objects Design Tasks
 - Adding an Assign Activity, 9-15
 - Adding an Invoke Activity, 9-12
 - Adding a Partner Link for File Adapter, 9-8
 - Creating a Partner Link, 9-7
 - Creating a SOA Composite Application with BPEL Process, 9-6

C

- Composite Services
 - download, 10-2
 - modify, 10-3
 - overview, 10-1
 - view, 10-2
- connection information
 - Application Server Connection, B-1
- create and upload custom interfaces
 - business events, 11-16
 - composite service validation, 11-14
 - creation, 11-6, 11-11
 - View and Administer composite services, 11-

15

- create and use custom interfaces
 - create steps, 11-2
 - overview, 11-1
 - use custom interfaces, 11-24
- create custom interfaces
 - composite services, 11-10
 - interface types, 11-2
- Creating BPEL Using Business Events
 - Assign, 6-20
 - Create a New BPEL Project, 6-3
 - Create a Partner Link AQ Adapter, 6-5
 - Create a Partner Link File Adapter, 6-13
 - invoke, 6-19
 - receive, 6-11

D

- Defining REST Service Invocation Metadata
 - Creating a Local Subscription to Invoke the REST Service, 13-13
 - Creating an Error Subscription to Enable Error Processing for REST, 13-16
 - Creating a Receive Event for REST Response (Optional), 13-18
 - Creating a Receive Event Subscription for REST Response (Optional), 13-19
 - Creating a REST Service Invoker Business Event, 13-12
- deploy and test bpel
 - deploy bpel, 9-22
 - test bpel, 9-24
- Deploy and Test Concurrent Program

- deploy bpel, 7-22, 7-23
- Deploy and Test Custom BPEL
 - deploy bpel, 11-39
 - test bpel, 11-40
- Deploy and Test Event BPEL
 - deploy bpel, 6-22
 - test bpel, 6-23
- Deploy and Test PL/SQL BPEL
 - deploy bpel, 3-38, 3-62
 - test bpel, 3-42, 3-65
- Discovering and Viewing Integration Interfaces
 - Generating Web Services, 2-6
 - overview, 2-1
 - REST Messages, 2-43
 - review details, 2-5
 - review WADL details, 2-18
 - review WSDL details, 2-8
 - search and view interfaces, 2-1
 - SOAP Messages, 2-24

I

- Integration Repository Annotation Standards
 - annotation glossary, A-120
 - business entity, A-41
 - business event, A-35
 - composite service - BPEL, A-115
 - concurrent program, A-23
 - guidelines, A-1
 - Java, A-4
 - PL/SQL, A-17
 - XML Gateway, A-25
- Invoke SOAP service
 - example, 12-33
- Invoke Web Services
 - Calling Back to Oracle E-Business Suite With SOAP Service Responses, 12-5
- Invoking a custom Java Bean Service
 - Creating and Compiling Custom Java APIs, 4-17
 - Creating a Security Grant, 4-37
 - Deploying a Custom Java Bean Service, 4-36
 - Deploying Custom Java Classes and Source Files, 4-34
 - Uploading a Custom Java Bean Service, 4-34
- Invoking a Java Bean Service
 - Recording the WADL, 4-38

- Invoking an Application Module Service Using Token Based Authentication and XML Payload
 - Creating a Project with a Java Class, 4-53
 - Creating a Security Grant, 4-52
 - Deploying a REST Service, 4-49
 - Invoking a REST Service Using a Java Class, 4-60
 - Recording the Deployed WADL URL, 4-51
- Invoking a REST Service
 - Creating a Project with a Java Class, 4-8
 - Deploying a REST Service, 4-2
 - Recording the WADL, 4-4, 4-6
- Invoking REST Services Using Service Invocation Framework
 - Defining REST Service Invocation Metadata, 13-11
- Invoking SOAP service steps
 - creating a Local Event Subscription for SOAP, 12-19
 - creating an Error Event Subscription for SOAP, 12-21
 - Creating a receive Event for SOAP Response, 12-22
 - Creating a receive Event Subscription for SOAP Response, 12-24
 - Creating Invoke and Receive Events, 12-17
- Invoking SOAP Services Using Service Invocation Framework
 - metadata definition, 12-16
 - understanding SOAP metadata definition, 12-12

J

- JSON Payload with REST Header
 - Deploying a PL/SQL REST Web Service, 3-81
 - Invoking REST Service Using a Java Client design time, 3-84
 - Invoking REST Service Using a Java Client runtime, 3-95
 - Recording the Deployed WADL URL, 3-83

O

- Oracle E-Business Suite Integrated SOA Gateway
 - component features, 1-2
 - Major Features, 1-1
 - Overview, 1-1

R

REST Extensibility

- postInvokeService, 13-30
- preInvokeService, 13-30

S

Sample Payload

- Inbound Purchase Order, C-3
- Supplier Ship and Debit Request, C-1

SOAP Extensibility

- addCustomSOAPHeaders, 12-45
- addWSSecurityHeader, 12-42
- postInvokeService, 12-42
- preInvokeService, 12-42
- setInputParts, 12-43

SOAP service invocation

- consideration, 12-48
- Extending Seeded Java Rule Function, 12-41
- Testing SOAP Service Invocation, 12-29
- Troubleshooting SOAP Service Invocation Failure, 12-35

T

Testing SOAP Service Invocation

- Command Lines, 12-32
- Test Business Event Page, 12-29
- Troubleshooting SOAP Service Invocation Failure
 - Concurrent Manager (CM) Tier JVM, 12-39
 - OACORE WebLogic Server, 12-36
 - Standalone JVM, 12-41

U

Understanding REST Service Invocation

Metadata

- Advanced Configuration, 13-9
- REST Service Details, 13-5

Understanding SOAP Messages

- SOA Header for XML Gateway Messages, 2-33
- SOAP Header for Applications Context, 2-30
- SOAP Messages Through SOA Provider, 2-37
- SOAP Security Header, 2-26

understanding SOAP metadata definition

Additional Subscription Parameters, 12-14

Invoke Web Service Wizard, 12-12

SOAP Service Security, 12-14

use custom interfaces

- design tasks, 11-25
- overview, 11-24
- runtime tasks, 11-39

Using Business Events

- deploy and test bpel, 6-21
- overview, 6-1
- using Business Events, 6-2

Using Business Service Objects

- deploy and test bpel, 9-22
- overview, 9-1
- using Business Service Objects REST Services, 9-25
- using Business Service Objects SOAP Services, 9-2
- using Business Service Objects WSDL design time, 9-5

Using Concurrent Program

- design tasks, 7-2
- Overview, 7-1
- runtime tasks, 7-21

Using Concurrent Program design tasks

- Adding a Partner Link for File Adapter, 7-8
- Assign activities, 7-15
- Creating a New BPEL Project, 7-6
- Creating a Partner Link, 7-7
- Invoke activities, 7-13

Using custom WSDL

- Add an Assign activity, 11-34
- Add an Invoke activity, 11-33

Using Custom WSDL

- Adding a Partner Link for File Adapter, 11-29
- Create a New BPEL Project, 11-27
- Create a Partner Link, 11-29

Using Java APIs as REST Services

- Application Module Service Invocation Example, 4-48

Using Java Bean Services

- Invoking a Custom Java Bean Service, 4-15
- Invoking a Custom Java Service from HTML Using JavaScript, 4-41
- Invoking a REST Service, 4-2
- Invoking a REST Service using a Java Class, 4-14

- overview, 4-1
- Using Open Interface Tables
 - Creating a Security Grant, 8-4
 - Deploying a REST Service, 8-2
 - Invoking a REST Service, 8-4
 - Overview, 8-1
 - Recording the WADL, 8-4
- Using Open Interface Tables and Views
 - Overview, 8-1
- Using PL/SQL
 - deploy and test bpel, 3-37, 3-61
 - overview, 3-1
 - Synchronous BPEL Process , 3-6, 3-46
 - using PL/SQL REST Service, 3-68
 - using PL/SQL WSDL, 3-2
- Using PL/SQL REST Service
 - JSON Payload, 3-79
 - XML Payload with REST Header, 3-69
 - XML Payload with REST Header runtime, 3-79
- Using PL/SQL WSDL
 - Add an Assign activity, 3-23
 - Add an Invoke activity, 3-19, 3-52
 - Add a Receive activity, 3-53
 - Add Assign activities, 3-55
 - Adding a Partner Link for File Adapter, 3-11, 3-51
 - Create a New SOA Composite Application with BPEL Process, 3-7, 3-49
 - Create a Partner Link, 3-10
- Using Service Invocation Framework to Invoke REST Services
 - Callback, 13-3
 - Managing REST Service Invocation Errors, 13-11
 - REST Service Security, 13-3
 - Understanding REST service invocation metadata, 13-4
- Using Service Invocation Framework to Invoke SOAP Services
 - overview, 12-1
- Using XML Gateway
 - deploy bpel, 5-51
 - overview, 5-1
 - test bpel, 5-53
 - using XML Gateway Inbound, 5-2
- using XML Gateway Inbound

- using XML Gateway Inbound design time, 5-2
- Using XML Gateway Inbound
 - Creating a New BPEL Project, 5-7
- Using XML Gateway Inbound by SOA Provider
 - Assign, 5-17
 - Creating a Partner Link, 5-11, 5-12
 - Invoke, 5-16
- Using XML Gateway Inbound SOA Provider
 - Runtime tasks, 5-22
- Using XML Gateway Inbound SOA Provider Run-Time Tasks
 - deploy, 5-22, 5-25
- Using XML Gateway outbound
 - using XML Gateway outbound, 5-30
- Using XML Gateway Outbound
 - deploy and test bpel, 5-50
- Using XML Gateway outbound design task
 - Add an Assign Activity, 5-49
 - Add an Invoke Activity, 5-48
 - Add a Partner Link for File Adapter, 5-46
 - Adding a Receive Activity, 5-44
 - create a new BPEL project, 5-34
 - create a Partner Link for AQ Adapter, 5-35
 - overview, 5-30

W

- Web Service Invocation Using SIF
 - invoking SOAP services, 12-25
 - message patterns, 12-3
 - Supporting WS-Security, 12-7
 - Web Service Input Message Parts, 12-7

X

- XML Payload with REST Header
 - Deploying a PL/SQL REST Web Service, 3-70
 - Invoking REST Service Using a Java Client, 3-73
 - Recording the Deployed WADL URL, 3-72